University of Cape Town

Masters Thesis

Using Neural Networks to identify Individual Animals from Photographs

Author:
Emmanuel Kabuga

Supervisor:
A/Prof Ian Durbach
Co-supervisors:
Dr. Bubacarr Bah
Mr Allan Clark

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Faculty of Science
Department of Statistical Sciences
Centre for Statistical in Ecology, the Environment, and Conservation

November 8, 2019

Declaration of Authorship

I, Emmanuel KABUGA, declare that this thesis titled, "Using Neural Networks to identify Individual Animals from Photographs" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed: Signed by candidate

Date: 08/11/2019

Abstract

Effective management needs to know sizes of animal populations. This can be accomplished in various ways, but a very popular way is mark-recapture studies. Mark-recapture studies need a way of telling if a captured animal has been previously seen. For traditional mark-recapture, this is achieved by applying a tag to the animal. For non-invasive mark-recapture methods which exploit photographs, there is no tag on the animal's body. As a result, these methods require animals to be individually identifiable. They assess if an animal has been caught before by examining photographs for animals which have individual-specific marks (Cross et al., 2014; Gomez et al., 2016; Beijbom et al., 2016; Körschens, Barz, and Denzler, 2018).

This study develops a model which can reliably match photographs of the same individual based on individual-specific marks. The model consists of two main parts, an object detection model, and a classifier which takes two photos as input and outputs a predicted probability that the pair is from the same individual (a match). The object detection model is a convolutional neural network (CNN) and the matching classifier is a special kind of CNN called a siamese network. The siamese network uses a pair of CNNs that share weights to summarise the images, followed by some dense layers which combine the summaries into measures of similarity which can be used to predict a match.

The model is tested on two case studies, humpback whales (HBWs) and western leopard toads (WLTs). The HBW dataset consists of images originally collected by various institutions across the globe and uploaded to the Happywhale platform which encourages scientists to identify individual mammals. HBWs can be identified by their fins and specials markings. There is lots of data for this problem. The WLT dataset consists of images collected by citizen scientists in South Africa. They were either uploaded to iSpot, a citizen science project which collects images or sent to the (WLT) project, a conservation project staffed by volunteers. WLTs can be identified by their unique spots. There is a little data for this problem. One part of this dataset consists of labelled individuals and another part is unlabelled.

The model was able to give good results for both HBWs and WLTs. In 95% of the cases the model managed to correctly identify if a pair of images is from the same HBW individual or not. It accurately identified if a pair of images is drawn from the same WLT individual or not in 87% of the cases. This study also assessed the effectiveness of the semi-supervised approach on the WLT unlabelled dataset. In this study, the semi-supervised approach has been partially successful. The model was able to identify new individuals and matches which were not identified before, but they were relatively few in numbers. Without an exhaustive check of the data, it is not clear whether this is due to the failure of the semi-supervised approach, or because there are not many matches in the data. After adding the newly identified and labelled individuals to the WLT labelled dataset, the model slightly improved its performance and correctly identified 89% of WLT pairs.

A number of computer-aided photo-matching algorithms have been proposed (Matthé et al., 2017). This study also assessed the performance of Wild-ID (Bolger et al., 2012), one of the commonly used photo-matching algorithm on both HBW and WLT datasets. The model developed in this thesis achieved very competitive results compared with Wild-ID. Model accuracies for the proposed siamese network were much higher than those returned by Wild-ID on the HBW dataset, and roughly the same on the WLT dataset.

Acknowledgements

Glory to the almighty God for the grace, mercy, strength, and the ability to achieve this work.

I would like to express my profound gratitude to my supervisors A/Prof Ian Durbach, Dr. Bubacarr Bah, and Mr Allan Clark. Despite multiple responsibilities, they accepted to supervise this thesis. Their guidance, availability, expertise, and insightful advice have been invaluable for the fulfilment of this thesis. They have been my role models for both the scientific career and life. It is a blessing to have worked under their supervision.

Acknowledgement to Dr. John Measey and Mr. Alex Rebelo for collecting and providing the toad photographs, and to the WLT volunteer group (toadnuts) in Noordhoek for taking Prof Ian Durbach and me out to gather photographs and see the toads in the wild.

A special acknowledgement goes to the African Institute for Mathematical Sciences for supporting and funding both my Structured and Research Masters, without this support this work would not have been achieved.

Thanks to the South African Centre for High Performance Computing (CHPC) platform for providing me with the computational resources.

Finally, I am extremely grateful to my family especially my elder brother Mr. Ananias BUCUMI for his continuous support, encouragement, and wise advice since my childhood.

Dedicated to my late parents
KIDURANYA Lazare
BAVUMIRAGIYE Jeanne

Contents

List of Figures

List of Abbreviations

DNN	**D**eep **N**eural **N**etwork
HBW	**H**umpback **W**hale
WLT	**W**estern **L**eopard **T**oad
CHPC	**C**entre for **H**igh **P**erformance **C**omputing

Chapter 1

Introduction

1.1 Background

1.1.1 Individual animal recognition in ecology

Effective management and conservation of wildlife rely on sound knowledge of population information (Caughley and Gunn, 1996). The knowledge of this information is crucial for addressing questions related to community, ecosystem function, population dynamics, and behavioural ecology (Schneider et al., 2019). For example assessing if a species faces extinction requires information on the existing demographics of the species. This information is obtained via studies that recognise individuals so that their life can be followed over time. This allows one to estimate their abundance and their survival rate (Lebreton et al., 1992). The commonly used technique to achieve individual species recognition is the use of invasive methods that apply a mark or a tag to the animal's body (Whitehead, Christal, and Tyack, 2000). These methods have been applied to both marine and terrestrial species of various sizes (Auckland, Debinski, and Clark, 2004; Watkins et al., 1993) to assess both theoretical and applied questions (Booth, 2004; Kohler and Turner, 2001). However, invasive procedures are expensive to implement and potentially reduce the animal's natural behaviour and performance (Wilson and McMahon, 2006), and disturb its activities and relationship to others (Cuthill, 1991). Tags applied to individuals do not last a lifetime and their loss and the non-reporting of retrieved tags can bias the estimation of population parameters (Bradshaw, Barker, and Davis, 2000; Schwarz and Seber, 1999). In addition to being impracticable with large populations, invasive methods cause ethical and welfare conflicts due to temporary or permanent application of tags (Wilson and McMahon, 2006; McMahon, Bradshaw, and Hays, 2006).

Alternatively, many species are dotted with patterns or special body markings such as spots, fins or tail shape that are unique and individual-specific (Arzoumanian, Holmberg, and Norman, 2005; Gamble, Ravela, and McGarigal, 2008) and can be referred to as natural marks. Body markings can be utilised for individual recognition. The methods which exploit natural markings are cost-effective and not harmful to an individual animal's life. The non-intrusive nature of these methods is advantageous for studying threatened and endangered animals (Kelly, 2001). These approaches have evolved as a reliable alternative to invasive methods and have been applied to a wide range of animals – mammals (Langtimm et al., 2004; Martínez-Jauregui et al., 2012), amphibians (Gamble, Ravela, and McGarigal, 2008), reptiles (Pellitteri-Rosa et al., 2010), and fishes (Speed, Meekan, and Bradshaw, 2007).

1.1.2 Computer vision for animal ecology

Deep learning algorithms are artificial intelligence algorithms capable of discovering and learning good representations from raw data using feature learning at different levels, with higher-level learned features expressed in terms of lower-level ones (Bengio, 2012a). Originally inspired from mammal visual cortex (Hubel and Wiesel, 1968), deep convolutional neural networks are a category of deep neural networks (Bengio, 2012a) comprising a sequence of layers of neurons, each one making use of convolution operations to extract the information from overlapping small regions from previous layers (Goodfellow, Bengio, and Courville, 2016). Deep learning techniques have improved the cutting edge computer algorithms in different domains – speech recognition (Hinton et al., 2012; Deng, Hinton, and Kingsbury, 2013), machine translation (Cho et al., 2014) and computer vision tasks including object recognition (He et al., 2016a; Simonyan and Zisserman, 2014a), object detection (Szegedy, Toshev, and Erhan, 2013), amongst others. However, deep learning tools and knowledge are still not common in ecology despite the exponential growth and complexity of ecological datasets (Christin, Hervet, and Lecomte, 2018).

Computer vision is referred to as a scientific field that uses computers to extract high-level understanding from digital images or videos with the perspective of automating tasks that can be performed by human visual system. (LeCun, Bengio, and Hinton, 2015). The growth of ecological computer vision evolved from different fields – computer science (Branson et al., 2014) and remote sensing (LaRue, Stapleton, and Anderson, 2017) amongst others. Computer vision can effectively improve the efficiency, reproducibility, and the accuracy of ecological image-based processing tasks via automation analysis (Dell et al., 2014; Pennekamp and Schtickzelle, 2013). The smallest unit in the image is a pixel denoting the intensity value in the visible spectrum. In the image, neighbouring pixels form a group identity based on pixels proximity, similarity, and orientation referred to as features. They are often locations of higher intensity in pixel values caused by objects of interest in the image like points, edges or the relationship with surrounding pixels. Changes in pixel values and the relationship with surrounding pixels among images produce a sequence of features. Based on features, computer vision algorithms are able to perform ecological image-based tasks that could be a laborious manual process (Weinstein, 2018).

The introduction of computer vision in ecology has improved the use of non-invasive techniques via image analysis. Computer vision algorithms assess the size, location, and characteristics of ecological objects of interest in images by means of features. Image features have been used to perform a range of tasks – ascertain the evolution of animal colouration, pattern signatures, size, and shape assessment (Stoddard, Kilner, and Town, 2014; Levy, Lerner, and Shashar, 2014). They have also been used to study animal camouflage via edge detection algorithms (Stoddard et al., 2016; Tankus and Yeshurun, 2008). They can be used to describe the size and the shape for both specimens and living animals (Olsen and Westneat, 2015; Lavy et al., 2015; Howland, Macfarlane, and Tyack, 2012).

One of the tasks performed by ecologists is estimating the size of a population in a given study area. The first step to achieve this task is to identify individuals. To perform individual-level identification, a computer vision algorithm makes use of the similarity of an image pattern to match different images. The matching algorithm extracts features from two images and computes the likelihood of these being from the same individual. For individuals with unique patterns or markings, this can be thought of as a cheaper way to avoid expensive trapping and tagging occasions that could be dangerous for both observers and animals. This approach has been applied to a wide range of animals such

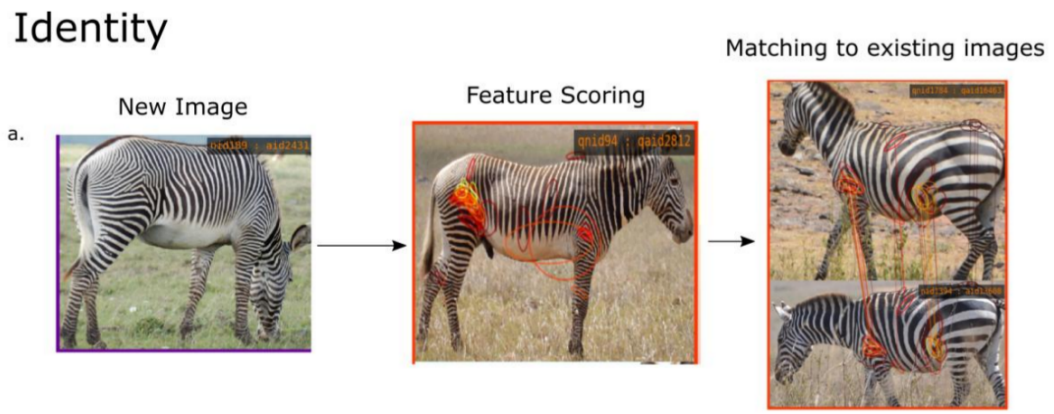


Figure 1.1: A matching algorithm scores the similarity of a photographed zebra to a library of known zebras to track individuals over time (Crall et al., 2013). The picture was taken from (Weinstein, 2018).

as zebras, elephants, and box turtles (Crall et al., 2013; Ardovini, Cinque, and Sangineto, 2008; Cross et al., 2014). This approach is even effective for animals with complex pattern or markings like catfish, giraffes, and whale sharks (Dala-Corte, Moschetta, and Becker, 2016; Bolger et al., 2012; Arzoumanian, Holmberg, and Norman, 2005). Earlier attempts made use of traditional machine learning algorithms with focus on feature-engineered or hard-coded feature extraction methods (Blanc, Lingrand, and Precioso, 2014; Lytle et al., 2010). Instead of hard-coded features, deep learning models are exposed to large amounts of data to learn particular features necessary for discriminating individuals (LeCun, Bengio, and Hinton, 2015). Deep learning models have significantly enhanced the model performance on predicting species-level identity for various animal taxa – mammal and coral (Gomez et al., 2016; Beijbom et al., 2016), fishes and butterflies (Villon et al., 2016; Hernández-Serna and Jiménez-Segura, 2014), and chimpanzee, lemur, and golden monkey identification (Deb et al., 2019; Körschens, Barz, and Denzler, 2018) amongst others. Figure 1.1 displays an example of individual identification performed on zebras based on their individual-specific lines. To match a new zebra against a library of known zebras, the new image passes through a algorithm which extracts the image key-points. Extracted key-points are tested against each library image's key-points to match the region of interest and compute the matching score. A library image with a high score is referred to as a match with the New Image.

1.2 Research objectives

The main goal of the thesis is to develop methods that can automatically match photos, and thus do individual identification. Even though it is possible to easily identify tens of individuals from photographs, the task of processing thousands of images periodically produced by citizens and scientists becomes tedious, time-consuming, and requires some automation. This thesis aims at creating a computer-based algorithm that can help automate the identification of species dotted with individual-specific marks. Individual identification results can be considered as an initial step which can be used to derive other useful measures such as estimation of the abundance/density and the survival rates of the species. These pieces of information can help the conservation policies and decision makers to keep track of species. This thesis has two case studies.

A) The western leopard toad (WLT) (*Sclerophrys pantherina*) is considered to be endangered by the International Union for Conservation of Nature as much of its habitat has been transformed. This large and charismatic amphibian has become the poster child of public conservation in Cape Town, and is even well known at a national level in South Africa. Much of the natural habitat of this species has been transformed into periurban areas, where the WLTs have adapted well to increases in breeding sites and manicured gardens. However, negotiating traffic on the way to and from breeding sites, leaves hundreds of animals dead every year, which has spurred the formation of citizen groups which task themselves with aiding WLTs cross roads (Measey et al., 2014). Some of the groups collect additional data including images of the WLTs for scientists to conduct capture-mark-recapture experiments, ideally with the aim of determining whether WLT populations are responding positively to citizen efforts. The WLTs have striking patterns on their backs that are unique to each WLT, and thus can provide input data for individual identification.

B) After hundreds of years of intense whaling, surviving humpback whales (HBWs) still experience difficulties to cope with the warming oceans as well as the daily fight against the fishing industry for food. Scientists study the marine life by monitoring ocean daily activities through camera surveillance systems. To contribute to HBW conservation effort, scientists use the shapes of HBW tail fins as well as unique markings found on their photographs to identify individuals they are dealing with. A large amount of this work continues to be done manually. Figure 1.2 displays example images for HBWs (first row) and WLTs (second row).

This thesis has the following specific goals:

1. The ultimate goal of this thesis is to build a machine learning algorithm which exploits individual-specific marks to automate the individual identification task from images. However, in some images, the salient object (animal) only occupies a little space, which makes the pattern or marks that are spots on the back of WLTs and shapes of HBW tail fins and special markings to not be clearly visible while they are the key feature of this thesis. Manually extracting salient objects from thousands of images seems to be tedious and time-consuming. Therefore, this project

   (a) firstly implements a semi-automated algorithm which utilises image thresholding and edge detection to generate bounding box data for a large number of training images. The process is faster than doing it manually.

   (b) uses generated bounding box data to train a convolutional neural network to fully automate the detection and extraction of the salient object from images. The resulting model is used to crop the remaining images around the salient object which were not accurately cropped by the semi-automated algorithm.

2. Use cropped images to train a matching classifier to individually identify HBWs and WLTs from photographs based respectively on shapes of HBW tail fins and special makings, and unique spots on the back of the WLTs.

3. Assess the effectiveness of a semi-supervised learning process for cases where there is only a small amount of labelled data. This is the situation faced with when constructing a model for WLTs.

4. Compare the model results with the results of the existing computer-aided photo-matching algorithm.

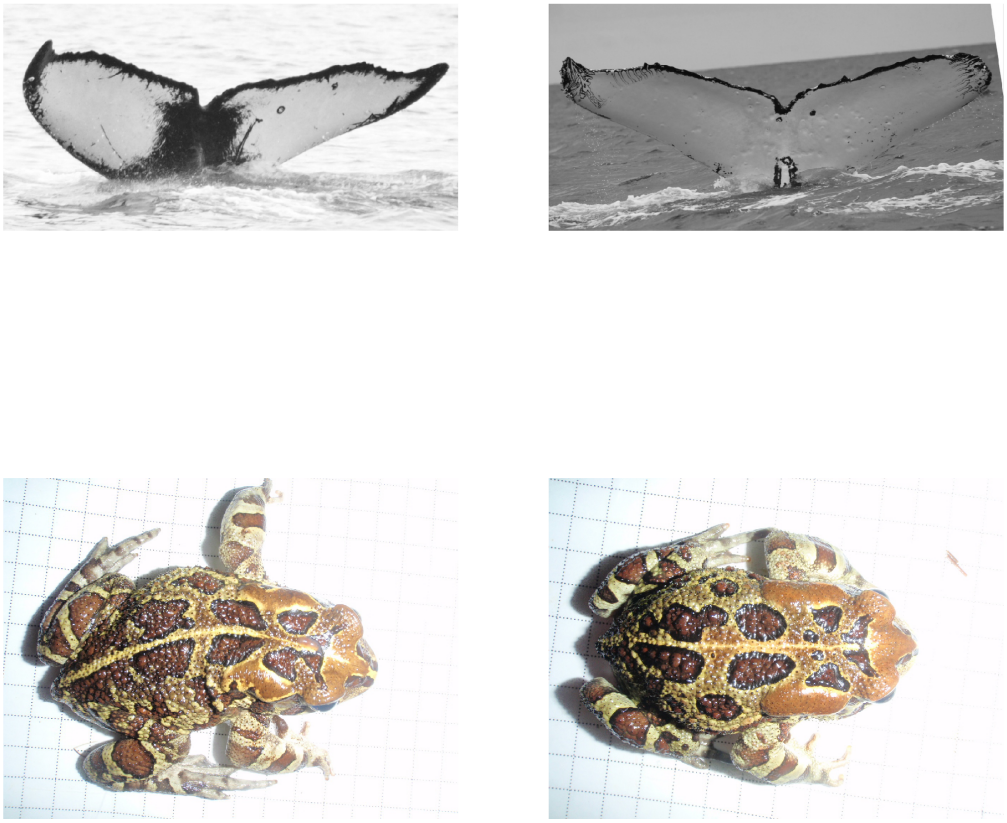


Figure 1.2: Example images for HBWs (first row) and WLTs (second row).

1.3 Outline of the thesis

This thesis comprises 5 chapters including this introduction chapter organised as follows: Chapter 2 starts off highlighting the theory behind the semi-supervised learning and then reviews the literature behind neural networks algorithms. Chapter 3 describes the data, presents methods used to obtain the bounding box data and train the automated bounding box detection models. Finally, it presents and discusses model results from both datasets. Chapter 4 presents the data, introduces the methods used to train the individual identification models, gives and discusses model results from both datasets including the semi-supervised process. The last chapter highlights some conclusions and suggestions for future work.

Chapter 2

Literature review

This chapter introduces some machine/deep learning concepts and their corresponding mathematical formulations. They will be invoked later on in the applications. It introduces the semi-supervised learning approach whose prime objective will be to utilise a model trained on a small dataset to identify new individuals in unlabelled data based on the matching probability between individual images. This chapter also highlights the siamese networks whose role is to compute the matching probability between pairs of individual images. However, to lay the groundwork for further understanding, deep neural networks (DNNs) and convolutional neural networks (CNNs) are introduced. Their building blocks are then used for model application architectures including the siamese networks. Deep learning models can over-fit to data if not carefully controlled. Hence, some regularisation strategies are introduced to deal with the over-fitting issues encountered in the training process. In the deep learning models, the learning process occurs by changing the model parameters over time to refine the computed functions for better predictions in the later iterations. Finally, optimisation algorithms whose main aim is to enable the learning process are reviewed.

2.1 Semi-supervised learning

Various machine learning algorithms are used to extract useful insights from the data. These algorithms can be categorised into supervised and unsupervised learning (Chapelle, Scholkopf, and Zien, 2009) algorithms based on the methodology used to learn from the data. This section highlights the semi-supervised learning (Chapelle, Scholkopf, and Zien, 2009; Zhu, 2005; Zhu and Goldberg, 2009) algorithm which lies between the aforementioned algorithms. The semi-supervised learning is motivated by the fact that most of data-driven real-world problems consist of a small amount of labelled data and abundant unlabelled data. Labelling real-world data is tedious, time-consuming and requires an expert annotator, so that even when data is plentiful, the proportion that is ultimately labelled may be small. The semi-supervised learning overcomes this bottleneck by utilising the model fitted on the labelled data to label the unlabelled data. The semi-supervised learning approach is an iterative process. Predictions in one step are added to training examples at the next step. This is done with both positive (predictions the model got right) and negative (predictions the model got wrong) examples.

2.1.1 Supervised learning

Supervised learning algorithms are supplied with correctly labelled data. They are pairs $\{\boldsymbol{i}^{(m)}, t^{(m)}\}_{m=1}^{M}$ of training instances $\boldsymbol{i}^{(m)}$'s together with their labels $t^{(m)}$'s. An instance $\boldsymbol{i}$ is a specific object often denoted by n-dimensional feature-vector $\boldsymbol{i} = [i_1, \ldots, i_n] \in \mathbb{R}^n$ with each dimension representing a particular feature. Instances are assumed to be drawn independent and identically distributed (iid) from an unknown underlying distribution $P(\boldsymbol{i})$. They are regarded as inputs to the learning process. Labels are observed values of instances, and reflect what kind of conclusions the algorithm could draw from the data.

Given the domain of instances I and its corresponding domain of labels T, assuming $P(\boldsymbol{i}, t)$ to be the underlying distribution on inputs and labels $I \times T$, given $\{\boldsymbol{i}^{(m)}, t^{(m)}\}_{m=1}^{M} \sim^{iid} P(\boldsymbol{i}, t)$, the supervised learning algorithm is challenged to learn the mapping $g : I \mapsto T$ in generalisable fashion such that $g(\boldsymbol{i})$ predicts the correct label t of a future instance i picked from the same distribution. Depending on the nature of the label t, the supervised learning can be further divided into two subgroups: classification and regression problems. The former is a supervised learning paradigm with categorical outcomes, the related function g is a classifier function. The latter deals with continuous outcomes, the corresponding function g is called the regression function. The best g, usually denoted by $g^{\star}$ is required to minimise the loss which determines the impact or the cost of producing a prediction $g(\boldsymbol{i})$ different from the observed value. The best g is mathematically given by $g^{\star} = \text{argmin}\, L(\boldsymbol{i}, t, g(\boldsymbol{i}))$, where $L(\boldsymbol{i}, t, g(\boldsymbol{i}))$ is the loss function.

Labelling the data involves a lot of effort in engineering features and creating labels, which result in generating less labelled data compared to the available unlabelled data. The next section introduces unsupervised learning which deals with unlabelled data.

2.1.2 Unsupervised learning

Unsupervised machine learning algorithms are furnished with the data and challenged to discover the underlying hidden patterns. They are trained on unlabelled data and aim to discover inherent similarities and group them into categories accordingly, and assign each category its own new label. Unsupervised learning is typically used for clustering and association applications.

2.1.3 Semi-supervised learning

The target of semi-supervised learning is to improve either the supervised or unsupervised learning algorithm by incorporation additional information in the learning process. The semi-supervised paradigm has various settings (Zhu and Goldberg, 2009) such as:

- **Constrained clustering**: It is viewed as the extension to the unsupervised clustering. Besides the unlabelled instances $\{\boldsymbol{i}^{(m)}\}_{m=1}^{M}$, the algorithm is provided with additional information about clusters such as a must-link constraint stating that two particular instances must belong to the same cluster, the cannot-link constraint asserting that two given instances cannot be in the same cluster or the constraint on the size of the cluster.
- **Semi-supervised classification**: It is the extension to the supervised classification setting. It deals with k labelled instances $\{\boldsymbol{i}^{(m)}, t^{(m)}\}_{m=1}^{k}$ and d unlabelled instances $\{\boldsymbol{i}^{(j)}\}_{j=k+1}^{k+d}$. Unlabelled instances are assumed to be more abundant than the

labelled ones, $d \gg k$. In this perspective, the goal is to train a classifier on the partially labelled instances that outperforms the one trained solely on labelled instances (Zhu and Goldberg, 2009). There is also another semi-supervised setting concerned with the regression with labelled and unlabelled instances. The setting is performed in the same fashion as the semi-supervised classification. This thesis focusses on the semi-supervised classification.

2.1.4 Transductive and inductive semi-supervised learning

Practically, there are two slightly different forms of semi-supervised learning known as inductive and transductive semi-supervised learning (Zhu and Goldberg, 2009; Chapelle, Scholkopf, and Zien, 2009). The inductive paradigm operates in the same way as the supervised learning algorithm. Its main goal is to predict the labels of future instances, in other words, the generalisation on unseen data. Since semi-supervised learning involves both labelled and unlabelled instances, the transductive paradigm aims at predicting labels on the unlabelled instances in the training data.

- **Inductive semi-supervised learning**: Given labelled instances $\{\boldsymbol{i}^{(m)}, t^{(m)}\}_{m=1}^{k}$ and unlabelled instances $\{\boldsymbol{i}^{(k)}\}_{j=k+1}^{k+d}$, the inductive learning aims at learning a mapping function $g : I \mapsto T$ in a way that g generalises well even on the future instances beyond $\{\boldsymbol{i}^{(k)}\}_{j=k+1}^{k+d}$, in the same way as the supervised learning approach.
- **Transductive learning**: Given respectively labelled and unlabelled instances $\{\boldsymbol{i}^{(m)}, t^{(m)}\}_{m=1}^{k}$, $\{\boldsymbol{i}^{(k)}\}_{j=k+1}^{k+d}$, transductive learning learns the mapping function $g : I^{k+d} \mapsto T^{k+d}$ such that g makes better predictions on unlabelled instances $\{\boldsymbol{i}^{(k)}\}_{j=k+1}^{k+d}$. The mapping is only defined on training data and is not required to generalise beyond this.

2.1.5 Self-training model

The key strategy behind this method is that it utilises its predictions to train itself, hence the self-training name. This kind of learning can be either inductive or transductive depending on the nature of the mapping function one is concerned with.

Given a set of labelled instances $\Psi = \{\boldsymbol{i}^{(m)}, t^{(m)}\}_{m=1}^{k}$ and unlabelled instances $\Omega = \{\boldsymbol{i}^{(k)}\}_{j=k+1}^{k+d}$, the self-training algorithm learns a mapping function g from labelled instances Ψ using a supervised learning algorithm. The learned function g is then utilised to make predictions on unlabelled instances Ω. A subset ϕ of unlabelled instances whose predictions have the highest probability of membership are removed from unlabelled instances and added to the labelled instances together with their predicted labels to increase the size of the training instances. The self-training model is re-trained and the whole process repeats again for a number of times.

2.2 Deep neural networks

DNNs are a class of machine learning algorithms originally inspired from the human brain (Rosenblatt, 1957; Widrow and Lehr, 1990; Rumelhart, Hinton, and Williams, 1985). They consist of a sequence of layers connected one to another whose components are processing units referred to as neurons. There are three categories of layers – input layer,

hidden layers, and output layer. The data is fed through the input layer and processed across hidden layers up to the output layer. The hidden and output layers are referred to as computation-performing layers (Aggarwal, 2018). However, the computations associated with hidden layers are not observed by the user (Nordhausen, 2009), hence the name hidden layers. For multilayer neural networks also known as feed-forward neural networks, layers are fully connected. Neurons between adjacent layers are fully pairwise linked. However, there is no connection between neurons within a layer (Agatonovic-Kustrin and Beresford, 2000). Each neuron receives the information from the previous layer, processes it, and sends the integrated information to neurons of the subsequent layer. The outcome from the output layer is the model prediction. A neural network with a single hidden layer is called a shallow neural network. The one with many hidden layers is referred to as a DNN.

DNN models involve three major components – data, parameters, and hyper-parameters. The data consists of the inputs and their corresponding observed values. The parameters are internal configuration variables to be learned by the model and they are required when making predictions. The hyper-parameters are external configuration variables set by the user. They help with the learning process to estimate the model parameters. Training a DNN model involves estimating a large number of parameters. There is no analytical solution for the weights. Hence, the model has to learn a set of parameters which allows for the minimisation of the error between observed and predicted values. Weights are randomly initialised and optimised through an iterative process. As a result, the learning process occurs through changing the parameters to allow the model improvement in future iterations.

The training process comprises two main phases, forward and backward propagation. Forward propagation is concerned with the computation of the predicted values (model predictions). Backward propagation or simply back propagation is the most commonly used technique when training DNNs. It was introduced in 1986 by and is still the gold standard in training DNNs models. It utilises the chain rule approach to propagate back the errors from the output layer to the input layer, hence the name back propagation. Its main purpose is to compute the gradients of the cost/error function with respect to the model parameters. These gradients show how changing the model parameters changes the cost function. In other words, they assess the correctness of the model parameters in response to predicted values. The cost function is the result of small errors accumulated throughout layers due to misspecification of the model parameters. For the sake of parameter correction, an optimisation algorithm is used to update parameters in the direction which minimises the cost function. Adjusting parameters over time refines the computed function so that it yields more accurate predictions.

This section describes the mathematical formulation behind DNNs. The rest of this section is organised as follows: To lay the groundwork for further understanding of DNNs, the logistic regression model is viewed as a neural network with a single neuron. Mathematical formulations are then generalised to multiple neurons in DNNs. The cost function and some common activation functions are introduced. The section concludes by describing the mathematics behind back propagation.

2.2.1 Logistic Regression as a neural network with a single neuron

The logistic regression model is one of the supervised learning algorithms used when the model output is binary, either 1 or 0 depending on whether the character of interest is reflected or not. It aims at categorising the input vector in one of the two categories

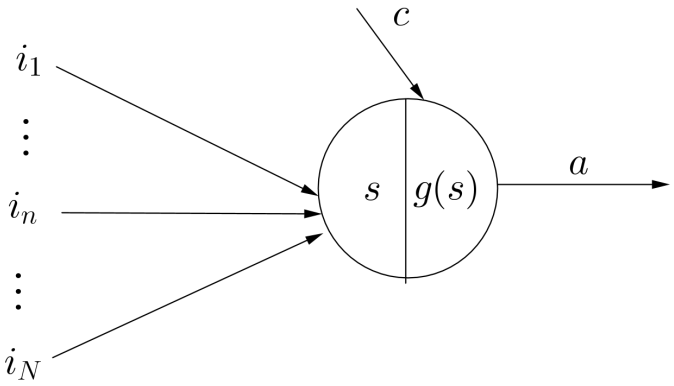


Figure 2.1: The neuron receives inputs $i_n\,(n = 1, 2, \ldots, N)$ from other neurons scaled by their corresponding weights γ_n. It computes the weighted sum s by adding a bias term c to their sum. An activation function $g(\cdot)$ is used to capture the non-linearity and send the output activation a to other neurons.

by assessing the probability of being categorised in the correct class. More precisely, the model prediction is a single probability. A decision is made by setting a threshold, usually 0.5. The model output is expected to be close to 1 if the character of interest is present and 0 otherwise. As a result, if the model output is beyond the threshold, the input vector is categorised in the class 1, otherwise 0. An artificial neuron involves three major processes (a) receiving the information, (b) processing it, (c) and sending out the processed information. Figure 2.1 summarises the three activities performed within a neuron.

Let $(\boldsymbol{i}, t)$ be a pair of input vector and its corresponding observed value, where $\boldsymbol{i} \in \mathbb{R}^N$ and $t \in [0, 1]$. The aim here is to classify the input vector in one of the two categories. The neuron receives the vector entries scaled with their corresponding weights and computes the weighted sum as follows:

$$s = \sum_{n=1}^{N} \gamma_n i_n + c, \tag{2.1}$$

where γ_n is the weight that links the n^{th} element of the vector to the neuron and c is known as the bias term. The parameters $\boldsymbol{\gamma}$ and c are used to control the model output. The weights are scaling factors that quantify the contribution of the vector elements to the predicted value whilst the bias shift the activation by adding a constant c to the input weighted sum. The relation (2.1) is vectorised as follows:

$$s = \boldsymbol{\gamma}^T \boldsymbol{i} + c, \tag{2.2}$$

where $\boldsymbol{\gamma}^T \in \mathbb{R}^N$ is a row vector whose components γ_n are individual weights associated with the components i_n of the vector $\boldsymbol{i}$.

Since one is interested in the probabilities, the output s of the linear function is not appropriate. Hence, the sigmoid non-linear function is used to transform the output of the linear function into probability. The model prediction is given by:

$$a = g(s) = \frac{1}{1 + e^{-(\boldsymbol{\gamma}^T \boldsymbol{i} + c)}}. \tag{2.3}$$

Given M training examples/vectors together with their observed values, their corresponding predicted values can be computed one after another using the same computations as described above. The input weighted sum associated with the m^{th} training example is given by:

$$s^{(m)} = \boldsymbol{\gamma}^T \boldsymbol{i}^{(m)} + c. \tag{2.4}$$

The computations done above process a single training example at a time. Instead of repeating the same scenarios M times, it is worth performing all computations at once by stacking together all input vectors into a matrix $\boldsymbol{I}$ whose columns are training examples. Its corresponding predictions are a vector $\boldsymbol{A}$ whose entries are individual predictions. This process is called vectorisation, it aims at using vectors and matrices wherever possible and tailored to speed up computations. This property is very important especially in deep learning where one deals with large amount of data and many parameters. The vectorised version corresponding to (2.4) is defined as follows:

$$\boldsymbol{S} = \boldsymbol{\gamma}^T \boldsymbol{I} + \boldsymbol{c}, \tag{2.5}$$

where $\boldsymbol{I}$ is a matrix whose columns are training samples and $\boldsymbol{S} \in \mathbb{R}^M$ is a row vector whose entries are defined in (2.4), and $\boldsymbol{c}$ is a row vector whose entries all equal to c.

The predictions associated with (2.5) are given by:

$$\boldsymbol{A} = g(\boldsymbol{S}) = \frac{1}{1 + e^{-(\boldsymbol{\gamma}^T \boldsymbol{I} + c)}}, \tag{2.6}$$

where $\boldsymbol{A} \in \mathbb{R}^M$ is a row vector whose entries are individual predictions given by (2.3). All the computations performed above correspond to a single neuron. The following section generalises the same computations to all neurons within a layer. The main aim of this section is to compute the model predictions via a DNN forward propagation.

2.2.2 DNN forward propagation

A DNN is a neural network with many layers. Neurons of consecutive layers are fully connected. However, neurons within the same layer are not connected. The information is processed from the input layer to the output layer. The output of each layer is the input to the next layer. This process is referred to as the forward propagation. Its main objective is to compute the network predictions. Figure 2.2 displays the architecture of a DNN.

Let $a_k^{[h-1]}$, $\gamma_{jk}^{[h]}$, and $c_j^{[h]}$ denote respectively the output activation associated with the k^{th} neuron in layer $h-1$, the weight linking the k^{th} neuron from layer $h-1$ to the j^{th} neuron in layer h and the bias corresponding to the j^{th} neuron in layer h. The input weighted sum to the j^{th} neuron is defined as follows:

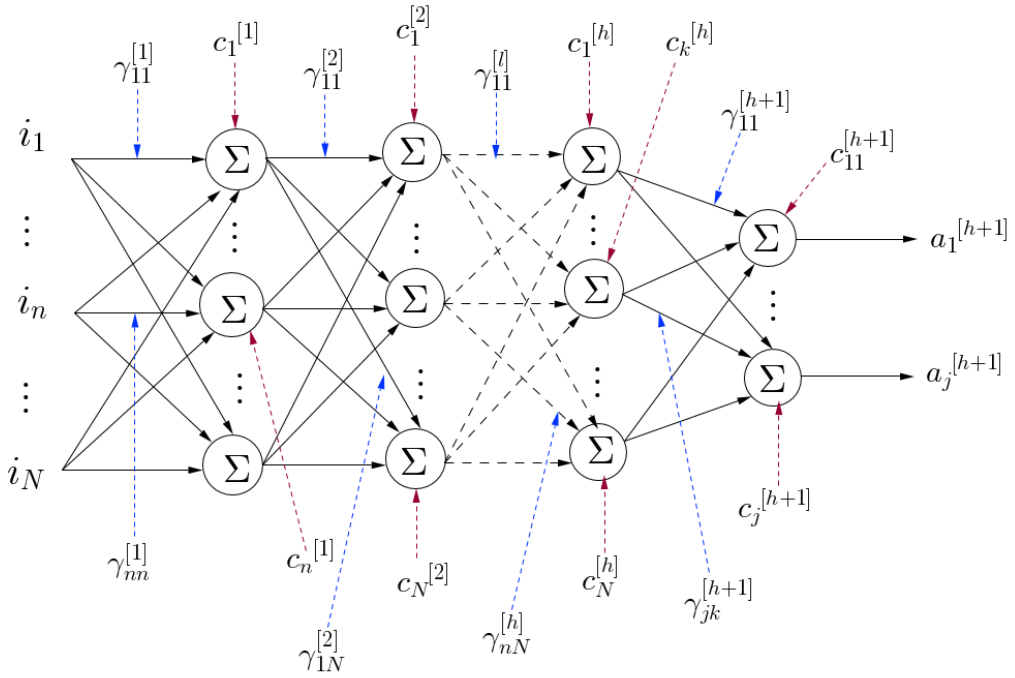


Figure 2.2: DNN architecture where $[i_1, \ldots, i_N]$ is the input vector, $\gamma_{jk}^{[h+1]}$ is the weight linking the k^{th} neuron from layer h to the j^{th} neuron in layer $h+1$, $c_j^{[h+1]}$ is the bias term corresponding to the j^{th} neuron in layer $h+1$, and $a_j^{[h+1]}$ is the output activation from the j^{th} neuron in layer $h+1$.

$$s_j^{[h]} = \sum_{k=1}^{K} \gamma_{jk}^{[h]} a_k^{[h-1]} + c_j^{[h]}, \tag{2.7}$$

where K is the number of neurons in layer $h-1$. The summation over k means that the input to the j^{th} neuron in layer h is the sum of individual output activations of all neurons in layer $h-1$ scaled by the weights linking them to the j^{th} neuron in layer h plus the bias term related to the j^{th} neuron in layer h.

The expression above is a sum of linear functions. The output of any sequence of linear operations can be written as the output of a single linear operation, and thus stacking linear layers on top of one another is not expected to give an improvement over what a single layer can do. Linear functions are also limited in the range of phenomena they can capture. To incorporate non-linearity, the output of the linear function is fed in the non-linear function. As a result, the output activation function corresponding to neuron j in layer h is given by:

$$a_j^{[h]} = g\left(s_j^{[h]}\right) = g\left(\sum_{k=1}^{K} \gamma_{jk}^{[h]} a_k^{[h-1]} + c_j^{[h]}\right), \tag{2.8}$$

where $g(\cdot)$ is a non-linear function.

Rather than process each neuron individually, all the information coming from all neurons in layer $h-1$ to all neurons in layer h can be performed at once. For all neurons K, the

vectorised form of (2.7) is written as follows:

$$\boldsymbol{s}^{[h]} = \boldsymbol{\gamma}^{[h]}\boldsymbol{a}^{[h-1]} + \boldsymbol{c}^{[h]}, \tag{2.9}$$

where $\boldsymbol{\gamma}^{[h]}$ is a matrix of weights whose row entries are weights linking the K neurons from layer $h-1$ to a single neuron in layer h, $\boldsymbol{s}^{[h]}$ is a vector whose entries are individual neuron input weighted sum in layer h, $\boldsymbol{a}^{[h-1]}$ the neuron output activations associated with the K neurons in layer $h-1$, and $\boldsymbol{c}^{[h]}$ is vector whose entries correspond to individual neuron biases in layer h. The output activations associated with all neurons in layer h are computed via the vectorised form related to (2.8) as follows:

$$\boldsymbol{a}^{[h]} = g\left(\boldsymbol{s}^{[h]}\right) = g\left(\boldsymbol{\gamma}^{[h]}\boldsymbol{a}^{[h-1]} + \boldsymbol{c}^{[h]}\right), \tag{2.10}$$

where $\boldsymbol{a}^{[h]}$ is a vector whose components are neuron individual output activations from layer h. These operations can be vectorised for computational efficiency as described in the previous section to process all training examples at once. Deep learning literature encompasses a number of activation functions. The following section highlights the commonly used ones.

2.2.3 Activation functions

When building a neural network, one of the choices to make is what kind of activation function to use across hidden layers and at the output layer. However, a number of activation functions are used in practice. The key feature for an activation function is for it to be differentiable. The following are some of the activation functions encountered in deep learning literature:

Sigmoid, $a = 1/(1+e^{-s})$, hyperbolic tangent, defined as $a = (e^{s} - e^{-s})/(e^{s} + e^{-s})$. The main downside of both activations is that when the value of s is either large or small the gradient is close to zero. This substantially slows down the optimisation algorithm.

One of the most used activation function in deep learning is the rectified linear unit commonly known as ReLU (Nair and Hinton, 2010; Glorot, Bordes, and Bengio, 2011) defined as $a = \max(0, s)$. One disadvantage of the ReLU is that its gradient is zero when s is negative. There is another modified version called Leaky ReLU (Xu et al., 2015) which overcomes this drawback. Instead of taking the gradient of zero when s is negative, it takes a small value that enables the gradient not to be zero. Its mathematical form looks like $a = \max(0.01s, s)$. The advantages of both ReLU/Leaky ReLU is that the gradient is positive and this accelerates the learning process.

Necessary tools to compute the model predictions are so far covered. Nevertheless, one needs to know how well the model is doing when compared to the observed values. The next section derives the cost function tailored to evaluate the error between the predicted and observed values.

2.2.4 Derivation of the cost function

To assess how well the predicted values are getting closer to the observed values each time the model is exposed to data, an error function known as the loss function is computed to measure the discrepancy between each predicted value and its corresponding observed

value. The cost function is referred to as the average loss from all predicted values. This section derives the binary cross entropy loss utilised when the output of the model is binary, either 0 or 1. Let t and $\hat{t}$ be respectively the observed and predicted values associated with a training example $\boldsymbol{i}$. The probability that $\boldsymbol{i}$ belongs to class 1 is given by:

$$Pr\left(t=1|\boldsymbol{i}\right)=\hat{t}. \tag{2.11}$$

Conversely, since probabilities sum to one, the probability that the same $\boldsymbol{i}$ belongs to class 0 is given by:

$$Pr\left(t=0|\boldsymbol{i}\right)=1-\hat{t}. \tag{2.12}$$

Both cases can be summarised in the following single equation:

$$Pr(t|\boldsymbol{i})=\hat{t}^{t}\left(1-\hat{t}\right)^{1-t}.$$

Applying the natural logarithm function on both sides yields:

$$\begin{aligned}\log Pr(t|\boldsymbol{i}) &= t\log\hat{t}+(1-t)\log\left(1-\hat{t}\right),\\ &= -L(\hat{t},t).\end{aligned} \tag{2.13}$$

The expression above is the negative of the loss function. The loss function $L(\hat{t},t)$ assesses the discrepancy between the predicted value $\hat{t}$ and the observed value t for a single training example. Given M inputs $\left\{(\boldsymbol{i}^{(m)},t^{(m)})\right\}_{m=1}^{M}$ the loss function associated with the m^{th} training example is yielded by:

$$L(\hat{t}^{(m)},t^{(m)})=-\left[t^{(m)}\log\hat{t}^{(m)}+(1-t^{(m)})\log\left(1-\hat{t}^{(m)}\right)\right]. \tag{2.14}$$

When training examples are independently and identically distributed, the probability associated with all training labels is the product of the probabilities of all training examples.

$$Pr(\text{M labels in training set})=\prod_{m=1}^{M}Pr(t^{(m)}|\boldsymbol{i}^{(m)}). \tag{2.15}$$

Since predicted values are desired to be closer to observed values, the maximum likelihood estimation is carried out to find parameters which maximise the probability of the training examples being in the correct class. However, maximising the probability is the same as maximising the log of the probability. As a result, using (2.13), the equation (2.15) can be simplified to

$$\log Pr(\text{M labels in training set})=-\sum_{m=1}^{M}L(\hat{t}^{(m)},t^{(m)}). \tag{2.16}$$

The cost function is an average loss function from all training examples. Maximising the log of the probability leads to minimising the cost function. The training aims at finding the parameters $\boldsymbol{\gamma}$ and $\boldsymbol{c}$ which minimise the overall cost function defined as follows:

$$E(\boldsymbol{\gamma}, \boldsymbol{c}) = \frac{1}{M}\sum_{m=1}^{M} L(\hat{t}^{(m)}, t^{(m)}) = -\frac{1}{M}\sum_{m=1}^{M}\left[t^{(m)}\log\hat{t}^{(m)} + (1-t^{(m)})\log(1-\hat{t}^{(m)})\right], \tag{2.17}$$

where $\hat{t} = \mathbf{a}^{[h]} = g\left(\boldsymbol{\gamma}^{[h]}\mathbf{a}^{[h-1]} + \mathbf{c}^{[h]}\right)$ is the output activation in the last layer.

In response to the cost function, the model may need to be refined by adjusting parameters. The next section introduces the back propagation process used to compute the gradients of the cost function in order to update the parameters in the direction which minimises the cost function.

2.2.5 DNN back propagation

The main goal of this section is to compute the derivatives of the cost function with respect to the weights and biases, $\frac{\partial E}{\partial \gamma_{jk}^{[h]}}$ and $\frac{\partial E}{\partial c_j^{[h]}}$. To achieve this, an intermediate quantity $\delta_j^{[h]}$ referred to as the error emanating from the neuron j in layer h is introduced. The rate of change of the cost function with respect to the parameters is then expressed in terms of the error term $\delta_j^{[h]}$.

The expression of the error term in layer h: This section computes the error originating from layer h in the neural network. Computations are generalised from a single neuron j to all neurons J in layer for the sake of vectorised implementations. Let the error related to neuron j in layer h be defined as follows:

$$\delta_j^{[h]} = \frac{\partial E}{\partial s_j^{[h]}}. \tag{2.18}$$

By means of the chain rule the relation above can be re-expressed in terms of the partial derivatives with respect to the output activations in layer h as follows:

$$\begin{aligned}\delta_j^{[h]} &= \sum_{k}^{K}\frac{\partial E}{\partial a_k^{[h]}}\frac{\partial a_k^{[h]}}{\partial s_j^{[h]}}, \\ &= \sum_{k}^{K}\frac{\partial E}{\partial a_k^{[h]}}\frac{\partial g\left(s_k^{[h]}\right)}{\partial s_j^{[h]}},\end{aligned} \tag{2.19}$$

where K is the number of neurons in layer h. After expanding the summation on the right hand side and differentiating, all terms vanish except one where $k = j$. Therefore, (2.19) simplifies to:

$$\delta_j^{[h]} = \frac{\partial E}{\partial a_j^{[h]}} g'\left(s_j^{[h]}\right). \tag{2.20}$$

The relation (2.20) expresses a component-wise error. The errors resulting from all neurons in layer h can be stacked together in a single vector for the sake of vectorisation. The vectorised expression can be written as follows:

$$\boldsymbol{\delta}^{[h]} = \frac{\partial E}{\partial \boldsymbol{a}^{[h]}} g'(\boldsymbol{s}^{[h]}), \tag{2.21}$$

where $\boldsymbol{\delta}^{[h]}$ is a vector whose components are error terms $\delta_j^{[h]}$ from individual neurons.

It is worth mentioning that, if h is the final layer, the expression of the error term is given by (2.21). Otherwise, its corresponding error will be expressed in terms the error from subsequent layers since back propagation sends the error from the last to the first layer. As a result, the following section establishes the expression of the error term across hidden layers.

Error propagation between consecutive layers: The error in a given layer is the result of accumulated errors throughout previous layers. Subsequently, during back propagation, the error is propagated layer to layer starting at the last layer. This section applies the chain rule technique to establish the relationship between errors of two consecutive layers.

The error in layer $h+1$ is given by the derivative of the cost function with respect to the input weighted sum in the same layer. Following (2.18), it can be written as follows:

$$\delta_k^{[h+1]} = \frac{\partial E}{\partial s_k^{[h+1]}}. \tag{2.22}$$

Applying the chain rule to the relation (2.18) to incorporate the term $s_k^{(h+1)}$ conveying model parameters in layer $h+1$, yields the following expression:

$$\delta_j^{[h]} = \sum_k^K \frac{\partial E}{\partial s_k^{[h+1]}} \frac{\partial s_k^{[h+1]}}{\partial s_j^{[h]}}. \tag{2.23}$$

Replacing (2.22) into (2.23) and swapping the two terms gives

$$\delta_j^{[h]} = \sum_k^K \frac{\partial s_k^{[h+1]}}{\partial s_j^{[h]}} \delta_k^{[h+1]}, \tag{2.24}$$

where the input weighted sum associated with the k^{th} neuron in layer $h+1$ is:

$$\begin{aligned} s_k^{[h+1]} &= \sum_{j'}^{J'} \gamma_{kj'}^{[h+1]} a_{j'}^{[h]} + c_k^{[h+1]}, \\ \frac{\partial s_k^{[h+1]}}{\partial s_j^{[h]}} &= \frac{\partial}{\partial s_j^{[h]}} \left(\sum_{j'}^{J'} \gamma_{kj'}^{[h+1]} g\left(s_{j'}^{[h]}\right) + c_k^{[h+1]} \right). \end{aligned} \tag{2.25}$$

After expanding (2.25) and differentiating, all terms become zero except one where $j' = j$. Subsequently, the expression above reduces to:

$$\frac{\partial s_k^{[h+1]}}{\partial s_j^{[h]}} = \gamma_{kj}^{[h+1]} g'\left(s_j^{[h]}\right) \tag{2.26}$$

After plugging (2.26) into (2.24), the error term corresponding to the j^{th} neuron in layer h expressed in terms of the error term in layer $h+1$ is given by:

$$\delta_j^{[h]} = \sum_k^K \gamma_{kj}^{[h+1]} \delta_j^{[h+1]} g'\left(s_j^{[h]}\right). \tag{2.27}$$

It is noticeable that in the forward computations, the summation is over the second index of γ_{kj}, simply meaning that the neuron k in layer $h+1$ receives the input weighted sum of all neuron output activations from layer h as input. In the backward process, the summation is over the first index, this simply notifies that an error that occurs in neuron j in layer h affects all the neurons in layer $h+1$ to which the neuron j sends connections. In other words when back propagating the error, the neuron j in layer h receives all errors from the K neurons in layer $h+1$.

The matrix notation associated with (2.27) is given by:

$$\boldsymbol{\delta}^{[h]} = \left(\boldsymbol{\gamma}^{[h+1]}\right)^T \boldsymbol{\delta}^{[h+1]} g'\left(\boldsymbol{s}^{[h]}\right), \tag{2.28}$$

where $\left(\boldsymbol{\gamma}^{[h+1]}\right)^T$ is the transpose of the weight matrix associated with layer $h+1$, $\boldsymbol{\delta}^{(h+1)}$ is a vector whose components $\delta_j^{[h+1]}$ are error terms originating from individual neurons in layer $h+1$, and $\boldsymbol{s}^{[h]}$ is a vector whose components are input weighted sum $s_j^{[h]}$ for each neuron in layer h.

Since weights link the previous layer to the next one, applying $\left(\boldsymbol{\gamma}^{[h+1]}\right)^T$ to the error $\boldsymbol{\delta}^{(h+1)}$ can intuitively be considered as moving the error backward from layer $h+1$ to layer h and yielding some kind of error measure at the output layer h. Finally multiplying by $g'\left(\boldsymbol{s}^{[h]}\right)$ moves the error backward through the output activation associated with layer h and producing the error $\boldsymbol{\delta}^{[h]}$ within the input weighted sum to layer h.

The relation (2.28) yields the expression of the error term across hidden layers. The following section utilises respectively the expression of the error term in the final layer and across hidden layers to respectively compute the gradient of the cost function with respect to parameters in the final and hidden layers.

The gradient of the cost function with respect to any parameter in the network: This section makes use of the error term introduced in the previous section to compute the rate of change of the cost function with respect to weights or biases. By means of the chain rule, the rate of change of the cost function with respect to bias $c_j^{[h]}$ corresponding to neuron j in layer h is yielded by:

$$\frac{\partial E}{\partial c_j^{[h]}} = \frac{\partial E}{\partial s_j^{[h]}} \frac{\partial s_j^{[h]}}{\partial c_j^{[h]}},$$

where the input weighted sum $s_j^{[h]}$ is defined as follows:

$$s_j^{[h]} = \sum_{k'=1}^{K'} \gamma_{jk'}^{[h]} a_{k'}^{[h-1]} + c_j^{[h]}. \tag{2.29}$$

After differentiating, the gradient of the cost function with respect to the bias in layer h is given by:

$$\frac{\partial E}{\partial c_j^{[h]}} = \delta_j^{[h]}. \tag{2.30}$$

This expression shows that the error corresponding to neuron j in layer h, is absolutely the same as the rate of change of the cost function with respect to the bias of the same neuron. By considering all neurons, the vectorised form is yielded by:

$$\frac{\partial E}{\partial \boldsymbol{c}^{[h]}} = \boldsymbol{\delta}^{[h]} \tag{2.31}$$

By using the chain rule, the rate of change of the cost function with respect to the weight $\gamma_{jk}^{[h]}$ can be re-expressed as follows:

$$\frac{\partial E}{\partial \gamma_{jk}^{[h]}} = \frac{\partial E}{\partial s_j^{[h]}} \frac{\partial s_j^{[h]}}{\partial \gamma_{jk}^{[h]}}. \tag{2.32}$$

After expanding further the summation on the right hand side of (2.29) and differentiating, all terms vanish except one for which $k = k'$. Therefore, the relation (2.32) simplifies to:

$$\frac{\partial E}{\partial \gamma_{jk}^{[h]}} = a_k^{[h-1]} \delta_j^{[h]}. \tag{2.33}$$

The rate of change of the cost function due to the weight $\gamma_{jk}^{[h]}$ is the product of input activation to the weight and the neuron output error term induced by the same weight.

The vectorised version is written as follows:

$$\frac{\partial E}{\partial \boldsymbol{\gamma}^{[h]}} = \boldsymbol{a}^{[h-1]} \boldsymbol{\delta}^{[h]}. \tag{2.34}$$

If h is the final layer, the gradient of the cost function with respect to biases and weights are respectively given by (2.31) and (2.34). Across hidden layers, the error term is given by (2.28). As a result, the gradients of the cost function with respect to biases and weights in the hidden layer h are respectively given by:

$$\frac{\partial E}{\partial \boldsymbol{c}^{[h]}} = \left(\boldsymbol{\gamma}^{[h+1]}\right)^T \boldsymbol{\delta}^{[h+1]} g'\left(\boldsymbol{s}^{[h]}\right), \tag{2.35}$$

and

$$\frac{\partial E}{\partial \boldsymbol{\gamma}^{[h]}} = \boldsymbol{a}^{[h-1]} \left(\left(\boldsymbol{\gamma}^{[h+1]} \right)^T \boldsymbol{\delta}^{[h+1]} g' \left(\boldsymbol{s}^{[h]} \right) \right). \tag{2.36}$$

DNNs considered so far take vectors as an inputs and utilise fully connected layers to summarise these input vectors and compute their corresponding predicted values. However, not only do fully connected layers use many parameters which make the model hard to train, but also in case of images, breaking an image into a vector of pixels can lead to the loss of certain neighbourhood properties. The next section introduces CNNs. They are a variant of DNNs which take 2-D input feature maps. They utilise convolutional operations and shared weights to summarise the input feature maps into vectors of feature representations. CNNs will be used later on in the application chapters to summarise images into vectors of feature representations before applying some fully connected layers to compute the predicted values.

2.3 Convolutional networks

CNNs are quite similar to ordinary neural networks and still follow the same process developed previously. Their inputs are 2-D (grayscale) or 3-D (RGB) images. They enable the use of convolutional operations. Weights are also arranged into 2-D or 3-D accordingly and referred to as filters/kernels. During the training, the model learns filters which detect some features rather that individual weights. Convolutions allow an efficient implementation of the forward propagation and a remarkable decrease of the number of parameters in the network. A CNN can be viewed as a sequence of connected layers. Each one transforms a volume of activations into another via a differentiable function. CNNs are good at learning abstractions or representations (Bengio, 2012a). Early layers might be learning edges, next layers some object's parts and later layers even more complex objects. There are three main categories of layers in convolutional networks – convolutional, pooling, and fully connected layers. The convolutional and fully connected layers have parameters to learn while pooling does not.

The convolution operation used in the deep learning literature does not match perfectly its definition in other fields such as engineering and pure mathematics (Goodfellow, Bengio, and Courville, 2016). Most of the deep learning libraries implement the cross-correlation which performs the dot product between the elements of the filter and the spatial location it overlaps without flipping the kernel. In deep learning the cross-correlation is referred to as a convolution.

2.3.1 Mathematical formulation of cross-correlation and convolution

Given a 2-D image Γ and a 2-D filter F of size $f_1 \times f_2$, the cross-correlation (Goodfellow, Bengio, and Courville, 2016) is formulated as follows:

$$(\Gamma \otimes F)_{(x,y)} = \sum_{p=0}^{f_1-1} \sum_{q=0}^{f_2-1} F_{(p,q)} \Gamma_{(x+p,y+q)}. \tag{2.37}$$

The cross-correlation slides the filter along the image dimensions and computes the dot product between the elements of the filter and the ones of the local position of the input feature map it overlaps.

The convolution corresponding to the same image Γ and the same filter F is obtained by flipping the filer F both horizontally and vertically (Kim and Casper, 2013) before correlating:

$$(\Gamma * F)_{(x,y)} = \sum_{p=0}^{f_1-1} \sum_{q=0}^{f_2-1} F_{(p,q)} \Gamma_{(x-p,y-q)}. \tag{2.38}$$

Both cross-correlation and convolution operations are demonstrated in the examples below. The first matrix is referred to as the filter and the second as a spatial location of the input feature map overlapped by the filter.

Cross-correlation:

$$\begin{bmatrix} k & b & g \\ a & e & m \\ n & u & l \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = k.1 + b.2 + g.3 + a.4 + e.5 + m.6 + n.7 + u.8 + l.9$$

The cross-correlation computes the component-wise dot product between the elements of the two matrices in the usual way.

Convolution:

$$\begin{bmatrix} k & b & g \\ a & e & m \\ n & u & l \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} l & u & n \\ m & e & a \\ g & b & k \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$
$$= l.1 + u.2 + n.3 + m.4 + e.5 + a.6 + g.7 + b.8 + k.9$$

Originally a convolution is performed by computing an inverted dot product between the elements of the kernel and the spatial location it overlaps. This means that the process starts off by the last element of the last row in the kernel and move from right to left whereas it starts off by the first element of the first row in the input feature map and move from left to right (first case in the above expression). However, the same result can be achieved by flipping the kernel horizontally and vertically before performing the cross-correlation operation (second case in the relation above).

It is worth mentioning that when training a convolutional network, if kernels are flipped before performing the forward pass, during back propagation, the δ's matrices are also flipped in the same way when updating the weights. If kernels are not flipped before performing the forward pass, only weights are flipped when performing the convolution to send the errors to the previous layer. It is noticeable that flipping horizontally and vertically simply means rotating the kernel weights by 180 degrees.

In terms of practical aspects, one can decide to compute either convolution or cross-correlation and the learning algorithm will learn the appropriate filter weights for the appropriate operation (Goodfellow, Bengio, and Courville, 2016). Most deep learning frameworks implement the cross-correlation as formulated in (2.37) and refer to it as a convolution. The same formulation is adopted for the rest of this section.

Convolution for RGB and grayscale images:

Let $\Gamma \in \mathbb{R}^{h \times w \times D}$ be an RGB image, $F \in \mathbb{R}^{f_1 \times f_2 \times D}$ a filter of weights , and $c \in \mathbb{R}$ the bias term. The corresponding convolution is defined as follows (T. Bolger et al., 2012; Lan et al., 2017; Ji et al., 2010):

$$G_{(x,y,z)} = (\Gamma \otimes F)_{(x,y,z)} = \sum_{p=0}^{f_1-1} \sum_{q=0}^{f_2-1} \sum_{d=0}^{D-1} F_{(p,q,d)} \Gamma_{(x+p,y+q,z+d)} + c, \tag{2.39}$$

where (x, y, z) denotes the 3-D coordinate of a pixel in an RGB image Γ and (p, q, d) the 3-D coordinate of a weight in the 3D filter F.

For grayscale images, where the number of channels is $D = 1$, the expression above simplifies to:

$$G'_{(x,y)} = (\Gamma \otimes F)_{(x,y)} = \sum_{p=0}^{f_1-1} \sum_{q=0}^{f_2-1} F_{(p,q)} \Gamma_{(x+p,y+q)} + c. \tag{2.40}$$

The subsequent sections only focus on the grayscale image case.

2.3.2 CNN forward propagation

In the forward process, a filter of weights is slid along the dimensions of the input feature map. At each location, a dot product between the elements of the filter and those of the input feature map it overlaps is computed. The resulting number represents a single pixel in the output feature map. The same process is repeated until all pixels of the input feature map are covered. The process ends up producing a 2-D feature map which yields the responses of the filter at each spatial position. The parameters associated with a convolutional layer are learnable filters. In the training process, the model will learn filters which activate some features such as edges, shapes and complex objects. However, utilising different filters to detect multiple features ends up outputting different 2-D feature maps which can be stacked into a single 3-D feature map. By convention, the feature map and filter are required to have the same number of channels.

The main advantage of using a convolutional layer over a fully connected layer is that it has a relatively small number of parameters. Hence, it is less prone to over-fitting. The reasons behind this are:

1. Parameter sharing: a low level feature detector (such as edge detector) or high level feature detector (such as object detector) which is useful in the region of an image can probably be useful in the other region of the same image. This means that a filter designed to detect a particular feature can be applied to different locations of the input feature map.

2. Local/sparsity connection: each point in the output feature map is only connected to a small location rather that the whole input feature map. In other words, each point in the output feature map is a linear combination of neighbouring points in the input feature map weighted by the filter weights.

3. Translation invariance: convolutional layers have the ability of preserving the image's identity or category despite changes in some specifics such as position or

appearance. An image is still recognised and assigned to the same category no matter some transformations such as rotation, shift, grayscaled, etc.

The expression of the convolution corresponding to the input feature map to layer h is defined as follows:

$$\begin{aligned} s^{[h]}_{(x,y)} &= \gamma^{[h]}_{(p,q)} \otimes a^{[h-1]}_{(x,y)} + c^{[h]}, \\ s^{[h]}_{(x,y)} &= \sum_{p=0}^{f_1-1} \sum_{q=0}^{f_2-1} \gamma^{[h]}_{(p,q)} a^{[h-1]}_{(x+p,y+q)} + c^{[h]}, \end{aligned} \tag{2.41}$$

where (x, y) is the coordinate of a pixel in the input feature map $a^{[h-1]}_{(x+p,y+q)}$ and (p, q) the coordinate of a weight in the filter $\gamma^{[h]}_{(p,q)}$. The summations over p and q show that any entry in the output depends on each weight entry in the filter.

$s^{[h]}_{(x,y)}$ is a linear combination of the inputs. As a result, a non-linear function is applied to incorporate the non-linearity. The mostly used activation function in the convolutional block is the ReLU activation function. The convolutional output activation in layer h is then given by:

$$a^{[h]}_{(x,y)} = g\left(s^{[h]}_{(x,y)}\right) = g\left(\sum_{p=0}^{f_1-1} \sum_{q=0}^{f_2-1} \gamma^{[h]}_{(p,q)} a^{[h-1]}_{(x+p,y+q)} + c^{[h]}\right), \tag{2.42}$$

where $g(\cdot)$ is the ReLU activation function.

The relations (2.41) and (2.42) can be used to make the prediction from any layer of the network.

Each time the convolution operator is applied, the image shrinks. For an image of size $h \times w$ convolved with an $f_1 \times f_2$ filter, the size of the output image is given by $(h - f_1 + 1) \times (w - f_2 + 1)$. Pixels in the corners of the input feature map are used much less in the output feature map compared to pixels in the middle. Thus, some useful pieces of information are thrown away. To avoid both scenarios, the zero-padding strategy is used. By convention, it adds some zero-pixels all around the image, enabling the control of the size of the output feature map.

Strided convolutions: The stride is the number of steps moved when convolving the image with the filter. For a $h \times w$ image convolved with an $f_1 \times f_2$ filter, with padding p and stride r, the image output size is $\lfloor \frac{h+2p-f_1}{r} + 1 \rfloor \times \lfloor \frac{w+2p-f_2}{r} + 1 \rfloor$. In case that the fraction is not an integer, the floor notation rounds it down to the nearest integer. This means that computations are only performed if the filter lies entirely within the image.

Even though it is possible to design reliable networks using only convolutional layers, most of neural network architectures comprise some pooling and fully connected layers.

Pooling layer: CNNs also make use of pooling layers which allow the decrease of the image size and subsequently the amount of parameters which leads to over-fitting. It speeds up computations and makes feature detectors more robust. The most common pooling layers are max pooling and averaging pooling. To apply max pooling an image is broken into different regions. The output from each region is the maximum in that region referred to as the winning unit.

The idea behind what max pooling is doing is that, if one considers regions as a set of features, a large number in the region represents a particular feature associated with that region. Hence, it deserves to be preserved in the output feature map.

Another type of pooling layer that is sometimes used in CNNs is average pooling. The process is the same as max pooling the only difference is that instead of considering the max in the given location, an average associated with all values in that location is considered.

The interesting property of a pooling layer is that it has some hyper-parameters such as the filter size and the stride. However, those hyper-parameters are fixed once either manually or by cross-validation. There are no parameters to learn during back propagation. The common choices of these parameters are $(f = 2, r = 2)$ which divides the height and the width of the image by 2, and the so-called overlapping pooling where $(f = 3, r = 2)$. Max pooling does not use padding and keeps the number of channels unchanged. The result of a max pooling layer applied to a $h \times w \times D$ image is $\lfloor \frac{h-f}{r} + 1 \rfloor \times \lfloor \frac{w-f}{r} + 1 \rfloor \times D$.

Though, there are no parameters to learn during back propagation, the error is propagated back through the pooling layer. In case of the max pooling, the error is assigned to the winning unit. Otherwise, for the average pooling, the error is divided by all entries of the pooling block and each one receives an equal number. CNNs end by putting the outputs of the convolutional layers through some fully connected layers.

2.3.3 CNN back propagation

The CNN back propagation applies the same formulations as the standard neural network. The key difference is that, in the forward propagation of CNNs, each pixel in the input feature map (layer h) only impacts a small number of pixels in the output feature map (layer $h + 1$). As a result, when back propagating, the error associated with a particular pixel in layer $h + 1$ is only propagated back to neighbouring pixels in layer h which contributed to its computation in the forward pass. Since filters of weights are used, their corresponding error terms δ are also matrices with the same dimensions. Their entries are individual error terms originating in a pairwise fashion from individual weights within the filter. Updating filters implies updating its individual weights. Hence, the prime goal of this section is to compute the gradients of the cost function with respect to individual weights within a filter and biases.

The rate of change of the cost function due to a weight in the filter: This section concerns with computing the gradient of the cost function $\frac{\partial E}{\partial \gamma_{(p',q')}}$, referred to as the measurement which assesses how changing the weight $\gamma_{(p',q')}$ in the filter changes the cost function.

In the forward propagation, each weight in the filter contributes to the dot product between the elements of the filter and the location of the input feature map it overlaps. This means that the weight $\gamma_{(p',q')}$ in the filter influences each element in the output feature map. The rate of change of the cost function due to the weight $\gamma_{(p',q')}$ in the filter is computed by means of the chain rule as follows:

$$\begin{aligned}\frac{\partial E}{\partial \gamma^{[h]}_{(p',q')}} &= \sum_{x=0}^{h-f_1}\sum_{y=0}^{w-f_2}\frac{\partial E}{\partial s^{[h]}_{(x,y)}}\frac{\partial s^{[h]}_{(x,y)}}{\partial \gamma^{[h]}_{(p',q')}}, \\ &= \sum_{x=0}^{h-f_1}\sum_{y=0}^{w-f_2}\delta^{[h]}_{(x,y)}\frac{\partial s^{[h]}_{(x,y)}}{\partial \gamma^{[h]}_{(p',q')}}, \end{aligned} \tag{2.43}$$

where the error term $\delta^{[h]}_{(x,y)} = \frac{\partial E}{\partial s^{[h]}_{(x,y)}}$.

Using (2.41) leads to

$$\frac{\partial s^{[h]}_{(x,y)}}{\partial \gamma^{[h]}_{(p',q')}} = \frac{\partial}{\partial \gamma^{[h]}_{(p',q')}}\left(\sum_{p=0}^{f_1-1}\sum_{q=0}^{f_2-1}\gamma^{[h]}_{(p,q)}a^{[h-1]}_{(x+p,y+q)} + c^{[h]}\right). \tag{2.44}$$

After expanding the summations on the right hand side and differentiating, all the components vanish except a single term where $p = p', q = q'$. As a result, the expression above simplifies to

$$\frac{\partial s^{[h]}_{(x,y)}}{\partial \gamma^{[h]}_{(p',q')}} = a^{[h-1]}_{(x+p',y+q')}. \tag{2.45}$$

After substituting (2.45) in (2.43), the gradient of the cost function with respect to any weight in the filter is given by:

$$\frac{\partial E}{\partial \gamma^{[h]}_{(p',q')}} = \sum_{x=0}^{h-f_1}\sum_{y=0}^{w-f_2}\delta^{[h]}_{(x,y)}a^{[h-1]}_{(x+p',y+q')}, \tag{2.46}$$

$$= \delta^{[h]}_{(x,y)} \otimes a^{[h-1]}_{(p',q')}, \tag{2.47}$$

where $\delta^{[h]}_{(x,y)} = \frac{\partial E}{\partial s_{(x,y)}}$. The two summations in the expression (2.46) originate from the weights sharing property. They put together all gradients corresponding to all outputs in layer h. The gradient of the cost function with respect to the bias term is given by:

$$\begin{aligned}\frac{\partial E}{\partial c^{[h]}} &= \sum_{x=0}^{h-f_1}\sum_{y=0}^{w-f_2}\frac{\partial E}{\partial s^{[h]}_{(x,y)}}\frac{\partial s^{[h]}_{(x,y)}}{\partial c^{[h]}}, \\ &= \sum_{x=0}^{h-f_1}\sum_{y=0}^{w-f_2}\frac{\partial E}{\partial s^{[h]}_{(x,y)}}\frac{\partial}{\partial c^{[h]}}\left(\sum_{p=0}^{f_1-1}\sum_{q=0}^{f_2-1}\gamma^{[h]}_{(p,q)}a^{[h-1]}_{(x+p,y+q)} + c^{[h]}\right), \end{aligned} \tag{2.48}$$

$$= \delta^{[h]}_{(x,y)} \tag{2.49}$$

The gradient of the cost function with respect to the bias term in layer h is exactly equal to the error term.

If h is the final layer, the expressions (2.47) and (2.49) respectively determine the gradients of the cost function with respect to the weights and biases. Otherwise, the error corresponding to the hidden layer h depends on the error from subsequent layers. Hence, the next section establishes the relationship between errors from consecutive layers.

Error propagation from layer $h+1$ to h : The main target is to express the error in layer h, $\delta^{[h]}_{(x,y)} = \frac{\partial E}{\partial s^{[h]}_{(x,y)}}$ in terms of the error of the subsequent layer. It can be noticed that changing a pixel value in the input feature map only affects a few elements in the output feature map. In other words, it only changes a small region π of the output feature map.

By means of the chain rule, the rate of change of the cost function E due to the change in the pixel $s_{(x,y)}$ is given by:

$$\begin{aligned}\frac{\partial E}{\partial s^{[h]}_{(x,y)}} &= \sum_{x'=0}^{f_1-1}\sum_{y'=0}^{f_2-1} \frac{\partial E}{\partial s^{[h+1]}_{(x',y')}} \frac{\partial s^{[h+1]}_{(x',y')}}{\partial s^{[h]}_{(x,y)}}, \\ &= \sum_{x'=0}^{f_1-1}\sum_{y'=0}^{f_2-1} \delta^{[h+1]}_{(x',y')} \frac{\partial s^{[h+1]}_{(x',y')}}{\partial s^{[h]}_{(x,y)}}, \end{aligned} \tag{2.50}$$

where (x', y') denotes the coordinate of a pixel in the region π within the feature map in layer $h+1$. Using (2.41) leads to:

$$\begin{aligned} s^{[h+1]}_{(x',y')} &= \sum_{p=0}^{f_1-1}\sum_{q=0}^{f_2-1} \gamma^{[h+1]}_{(p,q)} a^{[h]}_{(x'+p,y'+q)} + c^{[h+1]}, \\ &= \sum_{p=0}^{f_1-1}\sum_{q=0}^{f_2-1} \gamma^{[h+1]}_{(p,q)} g\left(s^{[h]}_{(x'+p,y'+q)}\right) + c^{[h+1]}. \end{aligned}$$

The second term on the right hand side of the expression (2.50) becomes:

$$\frac{\partial s^{[h+1]}_{(x',y')}}{\partial s^{[h]}_{(x,y)}} = \frac{\partial}{\partial s^{[h]}_{(x,y)}} \left(\sum_{p=0}^{f_1-1}\sum_{q=0}^{f_2-1} \gamma^{[h+1]}_{(p,q)} g\left(s^{[h]}_{(x'+p,y'+q)}\right) + c^{[h+1]} \right). \tag{2.51}$$

After expanding the dual summation on the right hand side and differentiating, all terms vanish except one where $x'+p=x$ and $y'+q=y$ that is $\gamma^{[h+1]}_{(p,q)} g\left(s^{[h]}_{(x,y)}\right)$. As a result, the expression (2.51) simplifies to:

$$\frac{\partial s^{[h+1]}_{(x',y')}}{\partial s^{[h]}_{(x,y)}} = \gamma^{[h+1]}_{(p,q)} g'\left(s^{[h]}_{(x,y)}\right). \tag{2.52}$$

Plugging back (2.52) into (2.50), the rate of change of the cost function due to changing a pixel in the input feature map is finally given by:

$$\frac{\partial E}{\partial s^{[h]}_{(x,y)}} = \sum_{x'=0}^{f_1-1}\sum_{y'=0}^{f_2-1} \delta^{[h+1]}_{(x',y')} \gamma^{[h+1]}_{(p,q)} g'\left(s^{[h]}_{(x,y)}\right). \tag{2.53}$$

From $x'+p=x$ and $y'+q=y$, it can be deduced that $p=x-x'$ and $q=y-y'$. Replacing p and q with their equivalent expressions yields:

$$\frac{\partial E}{\partial s^{[h]}_{(x,y)}} = \sum_{x'=0}^{f_1-1}\sum_{y'=0}^{f_2-1} \delta^{[h+1]}_{(x',y')} \gamma^{[h+1]}_{(x-x',y-y')} g'\left(s^{[h]}_{(x,y)}\right).$$

It is required to flip (rotate by 180 degrees) the filter weights when sending errors to previous layers. As a result, the above expression becomes:

$$\begin{aligned}
\frac{\partial E}{\partial s^{[h]}_{(x,y)}} &= \sum_{x'=0}^{f_1-1}\sum_{y'=0}^{f_2-1} \delta^{[h+1]}_{(x',y')} \left(\text{rot180}\left(\gamma^{[h+1]}_{(x-x',y-y')}\right)\right) g'\left(s^{[h]}_{(x,y)}\right), \\
&= \sum_{x'=0}^{f_1-1}\sum_{y'=0}^{f_2-1} \delta^{[h+1]}_{(x',y')} \gamma^{[h+1]}_{(x+x',y+y')} g'\left(s^{[h]}_{(x,y)}\right), \qquad (2.54) \\
\delta^{[h]}_{(x,y)} &= \delta^{[h+1]}_{(x',y')} \otimes \gamma^{[h+1]}_{(x,y)} g'\left(s^{[h]}_{(x,y)}\right), \qquad (2.55)
\end{aligned}$$

$\gamma^{[h+1]}_{(x,y)}$ is a set of filters whose weights connects the elements from layer h to $h+1$. Intuitively, convolving $\gamma^{[h+1]}_{(x,y)}$ with $\delta^{[h+1]}_{(x',y')}$ can be thought of sending the error backward from layer $h+1$ to h and producing some sort of error measure at the output layer h. Multiplying by $g'\left(s^{[h]}_{(x,y)}\right)$ propagate the error back through the output activation related to layer h and yielding the error $\delta^{[h]}_{(x,y)}$ in the input weighted sum of layer h.

After replacing $\delta^{[h]}_{(x,y)}$ by its expression in (2.47) and (2.49), the gradients of the cost function with respect to weights and biases in the hidden layer h are respectively yielded by:

$$\frac{\partial E}{\partial \gamma^{[h]}_{(p',q')}} = \left(\delta^{[h+1]}_{(x',y')} \otimes \gamma^{[h+1]}_{(x,y)} g'\left(s^{[h]}_{(x,y)}\right)\right) \otimes a^{[h-1]}_{(p',q')}. \tag{2.56}$$

and

$$\frac{\partial E}{\partial c^{[h]}_{(p',q')}} = \delta^{[h+1]}_{(x',y')} \otimes \gamma^{[h+1]}_{(x,y)} g'\left(s^{[h]}_{(x,y)}\right) \tag{2.57}$$

Preceding sections have covered the building blocks of a neural network architecture. The next section introduces a class of neural networks known as siamese neural networks. They are popular for tasks involving similarity learning between two objects such as images. To ensure that a pair of images is transformed in the same way, the siamese network makes use of two identical sub-networks, with same configuration and shared weights. Parameter

updating is replicated across the twin sub-networks. The siamese network architecture will be used in Chapter 4 to computer the matching probability between pairs of individual images.

2.4 Siamese neural networks

The usual methods such as neural networks and support vector machines used to perform a discriminative learning require in advance the knowledge of all possible categories as well as their corresponding training examples. Subsequently, these methods tend to perform poorly when faced with applications with a great number of categories where only a few training examples per category are available, or when only a subset of whole categories is provided. Tackling such kind of applications requires the use of methods that extract insights from the available data without making use of categories. A common method is performing a distance-based approach which computes the similarity metric between features or patterns extracted from objects to be classified or identified. The learned similarity metric can later generalise on even unseen categories. The type of neural networks which deals with this kind of tasks are the so-called siamese networks. They were first introduced in 1990s for fingerprint identification (Baldi and Chauvin, 1993) and signature verification (Bromley et al., 1994). Afterwards, they have been used for performing various tasks such as face verification and recognition (Chopra, Hadsell, and LeCun, 2005; "Deep face recognition."), different matching problems ranging from patch matching, stereo matching and aerial-to-ground image matching (Zagoruyko and Komodakis, 2015; Zbontar and LeCun, 2015; Lin et al., 2015) and one-shot learning problems (Koch, 2015) amongst others. A siamese network possesses two twin sub-networks with shared weights. It is exposed to both alike pairs (with similar objects) and unlike pairs (with different objects) and computes the similarity between them. The similarity is expected to be closer to 1 for a pair composed of same objects and closer to 0 otherwise. The similarity is derived by means of a standard metric or not. This section, highlights most common similarity metrics and different ways the similarity is computed from the model.

2.4.1 Common distance metrics for similarity assessment

Similarity measures are metrics used to quantify how much two objects are alike. The similarity score usually lies between 0 and 1. As a result, two identical objects are deemed to have a similarity of 1 and two completely different objects have 0 similarity. The similarity is the core component of algorithms such as recommendations systems, clustering, classification, and image retrieval. The similarity and the distance are two opposite quantities. The smaller the distance between two candidates, the higher the degree of similarity. This section introduces some common distance metrics amongst others encountered in the machine learning literature. Some of these will be used to compute different distances metrics used to predict the matching probability between individual images in Chapter 4.

a) **Euclidean distance**: is the most common distance metric. It produces the length of the path between two objects. The Euclidean distance between two vectors $\boldsymbol{v_r}$ and $\boldsymbol{v_s}$ in m-dimensional space is defined as follows:

$$d(\boldsymbol{v_r}, \boldsymbol{v_s})_E = ||\boldsymbol{v_r} - \boldsymbol{v_s}||_2 = \left[\sum_{k=1}^{m} (v_{rk} - v_{sk})^2\right]^{\frac{1}{2}}.$$

b) **Manhattan distance**: expressed distance in terms of the sum of the absolute values between data points of two instances. The Manhattan distance between two m-dimensional vectors, is given by:

$$d(\boldsymbol{v_r}, \boldsymbol{v_s})_M = ||\boldsymbol{v_r} - \boldsymbol{v_s}||_1 = \sum_{k=1}^{m} |v_{rk} - v_{sk}|.$$

c) **Minkowski distance**: is a generalisation of the Euclidean and the Manhattan distance. The Minkowski distance between two vectors in m-dimensional space is yielded by:

$$d(\boldsymbol{v_r}, \boldsymbol{v_s})_{MK} = ||\boldsymbol{v_r} - \boldsymbol{v_s}||_\alpha = \sum_{k=1}^{m} \left[|v_{rk} - v_{sk}|^\alpha\right]^{1/\alpha},$$

where α denotes the order of the Minkowski metric. Though the expression above is defined for any $\alpha > 0$, only special cases where $\alpha = 1$,2, and ∞ are encountered in practice. $\alpha = 1$ denotes the Manhattan distance also called L1-norm, $\alpha = 2$ the Euclidean distance also called L2-norm, and $\alpha = \infty$ corresponds to the Chebyshev distance also known as the Lmax-norm.

d) **Cosine similarity**: the core idea behind cosine similarity is the dot product also known as the inner product. It computes the normalised inner product between two vectors that is the value of the cosine of the angle between them. The cosine similarity between two vectors is defined as follows:

$$s(\boldsymbol{v_r}, \boldsymbol{v_s}) = \cos(\boldsymbol{v_r}, \boldsymbol{v_s}) = \frac{\boldsymbol{v_r}.\boldsymbol{v_s}}{||\boldsymbol{v_r}||||\boldsymbol{v_s}||} = \frac{\sum_{k=1}^{m} v_{rk} v_{sk}}{\left[(v_{rk})^2\right]^{1/2} \left[(v_{sk})^2\right]^{1/2}}$$

The cosine judgement is based on orientation rather than the magnitude. Hence, two vectors with identical orientation have a cosine similarity of 1 no matter their magnitude, orthogonal vectors have 0 cosine similarity, and diametrically opposed vectors have -1 cosine similarity. However, cosine similarity is usually used in the positive space where the output lies in the interval of 0 and 1.

The next section utilises some of the similarity metrics introduced in this section to compute the similarity between two images in the siamese network. Since images are higher dimensional objects, they are first reduced to feature-vector representations via a CNN. Extracted features are used to compute the similarity.

2.4.2 Siamese networks for learning the similarity between two objects

The input to the siamese network is a pair of objects to be compared. A siamese neural network utilises two convolutional sub-networks with shared weights to summarise a pair of objects into vectors of feature representations. Feature vectors are then compared using some standard distance metric. A siamese network encompasses two main properties (Koch, 2015):

a) **Prediction consistency**: shared weights guarantee that two images from the same object are mapped to the same receptor field in the feature space since both sub-networks compute the same function.

b) **Symmetry**: The joining layer computes the same metric inconsequential the twin network that processes the image.

Some approaches are observed when dealing with siamese networks such as learning the similarity metric from the data, inducing the distance metric from learned features, and learning the similarity without a distance metric.

2.4.3 Similarity metric learned from the data

The core idea behind this perspective is finding a function which maps the input space into target space such that the distance between a pair of objects in input space is approximated by a simple distance (like Euclidean distance) in the target space (Chopra, Hadsell, and LeCun, 2005). Given a differential function $g_\gamma(i)$ parametrised by γ, one is required to find the shared parameter γ so that the similarity metric $s_\gamma = ||g_\gamma(i^{(1)}) - g_\gamma(i^{(2)})||$ is minimised if $(i^{(1)}, i^{(2)})$ is a similar pair and large otherwise. $g_\gamma(i^{(1)})$ and $g_\gamma(i^{(2)})$ are higher level feature vector representations. The corresponding training loss function is the so-called contrastive loss (Chopra, Hadsell, and LeCun, 2005) defined as follows:

$$L(\gamma, (i^{(1)}, i^{(2)}, t)^k) = (1-t)\, L_s \left(s_\gamma \left(i^{(1)}, i^{(2)} \right)^k \right) + t L_d \left(s_\gamma \left(i^{(1)}, i^{(2)} \right)^k \right), \quad (2.58)$$

where t is a binary target which is either 0 if both objects are similar or 1 if different. $(i^{(1)}, i^{(2)}, t)^k$ denotes the k^{th} sample made of a pair of objects and their corresponding label. L_s and L_d are respectively partial loss function for alike pair (similar objects) and unlike pair (different objects). The loss function is designed in a way that minimising it result in the decrease of L_s and an increase of L_d.

2.4.4 Induced metric from learned features

This approach focusses on learning features representations and induces the distance metric from them later on. A standard neural network architecture which is replicated across the twins of the siamese network is used to extract most salient feature representations from raw pixels. A standard metric such as the Euclidean distance, the Manhattan distance or the cosine similarity is utilised to compute the similarity score between extracted features. Expressing similarity as a probability may be easier to interpret than if it is expressed as a score. Rather than computing the similarity score via a metric, an element-wise metric is performed on feature vectors to obtain a single vector of the same length as the original feature vectors. Sometimes, a normalisation layer such as mean-normalisation, L2-normalisation or max-normalisation (where each component in the feature vector is divided by the maximum) is performed before computing an element-wise metric (Taigman et al., 2014). Operations like element-wise dot product (Zagoruyko and Komodakis, 2015) or absolute value of element-wise difference between feature vectors (Koch, 2015) are used to compute the merge layer. The probability of the similarity is computed via a sigmoidal function either directly after the merge layer or after a fully connected neural network.

The output of the sigmoid function is a probability. Hence, the corresponding objective function is the binary cross-entropy loss defined as follows:

$$L\left(i^{(1)}, i^{(2)}, t\right)^k = t\left(i^{(1)}, i^{(2)}\right)^k \log p\left(i^{(1)}, i^{(2)}\right)^k + \left(1 - t\left(i^{(1)}, i^{(2)}\right)^k\right) \log\left(1 - p\left(i^{(1)}, i^{(2)}\right)^k\right). \quad (2.59)$$

The back propagation algorithm is used to learn the weights that minimise the loss function above. Since weights are shared in the siamese network, the parameter updating is mirrored across the twin networks.

2.4.5 Similarity learned from the data

This approach focusses on learning the similarity between objects without involving any standard distance metric. The key idea behind this method is extracting higher-level visual representations via a CNN. The two feature vectors are then concatenated in one feature vector whose length is twice long. These vectors are then utilised to train a visual similarity network whose output is the similarity between original images (Dosovitskiy et al., 2015; Zbontar and LeCun, 2015; Han et al., 2015). Standard metrics are deemed to be independent from the data at hand and hence might not be conveying the non-linear inner structure of visual representations (Garcia and Vogiatzis, 2017). Training deep learning models is challenging. It requires some experimentations and strategies. The next section introduces some of these challenges.

2.5 Challenges in deep learning models

This section highlights three common challenges faced when training deep learning models – under-fitting, over-fitting, and non-convergence issues. Under-fitting occurs when the algorithm fails to fit the data and over-fitting when it fails to generalise. Sometimes, it may happen that the optimisation algorithm fails to converge or takes a long time to reach the minimum. This issue is due to some regions where the gradients are zero or closer to zero.

2.5.1 Generalisation, capacity, under-fitting and over-fitting

What matters more in machine/deep learning is not how the algorithm performs on training data but how it does the job on previously unseen data. This ability is referred to as generalisation. During training, an error measure known as the training error is computed on a training set. The training process tries to decrease it through an optimisation process. What makes machine learning different from optimisation is that the generalisation error is also required to be small. The model generalisation is assessed by its performance on a test set which is an unobserved input data drawn separately from the training set. However, the training and test datasets are expected to come from the same underlying distribution to be able to relate the training error to the test error.

Factors assessing how good a model performs are its ability to make the training error and the gap between it and the test error small. These factors are related to two main challenges faced in machine/deep learning, known as under-fitting and over-fitting. The under-fitting issue happens when the model does not have the ability to get a sufficiently

small training error (high bias). Over-fitting occurs when the model fails to generalise, the gap between the training and test error is too high (high variance). Both cases can be controlled by altering the model capacity. Roughly speaking, the model capacity is the ability of fitting a wide range of functions. Models with low capacity are likely to under-fit, they struggle to fit the training set. Models with high capacity are likely to over-fit, they memorise the training set properties and fail to generalise. In general, models perform better when their capacity is proportional to the complexity of the task at hand and the amount of data provided. This means that, models with low capacity are not able to perform complex tasks and models with high capacity may over-fit when their capacity exceeds the task at hand.

2.5.2 Local optima, plateaus and saddle points

It used to be believed that the optimisation process of deep learning models gets stuck in the local minima (T. Bolger et al., 2012; Gori and Tesi, 1992). However the understanding of the local minima has changed through the advanced theory of the deep learning. When training a neural network most points with zero gradients in cost function are not local minima but saddle points instead (Dauphin et al., 2014; Choromanska et al., 2014). The intuition is that when the gradient is zero in each direction there is either a convex-like function where in all directions curves bend upwards or concave-like function where all curves bend downwards. In high dimensional space, it is more likely to get regions where some directions bend upwards and others downwards (saddle point) rather than have all directions bending downwards (local minimum). Saddle points are surrounded by plateaus where gradients are zero or closer to zero. Since the surface is almost flat when gradient is small, it substantially slows down the learning process and gives an illusory impression of the existence of the local minima. That is where the optimisation algorithms like momentum, RMSProp and Adam designed to speed up the learning process intervene. However, neural networks are complex optimisation problems. The understanding of their representations in the high dimensional space is still an area of research.

Different approaches are carried out to overcome the over-fitting issues in practice. The next section highlights the strategies used to avoid over-fitting when training deep learning models to enhance the model performance.

2.6 Regularisation

Various definitions of regularisation are encountered in machine learning literature. In traditional machine learning and optimisation, the term regularisation refers to the penalty term added to the loss function (Bishop and Others, 1995). Recently, Goodfellow, Bengio, and Courville (2016) defined regularisation as any modification introduced into the learning algorithm intended to decrease the generalisation error but not its training error. Kukačka, Golkov, and Cremers (2017) refer to regularisation as any strategy aiming at improving the model generalisation. This section introduces commonly used strategies to regularise bigger networks – norm penalties designed to reduce the complexity of the model, dropout referred to as ensemble model strategy to enhance the performance, data augmentation to account the limitation of training data, and early stopping to avoid over-fitting.

2.6.1 Norm penalties

Norm penalties add an extra term to the standard cost function to constrain the weights to remain relatively small. The regularised cost function takes the following form:

$$\mathcal{E}(\gamma, c) = E(\gamma, c) + \lambda R(\gamma), \tag{2.60}$$

where $E(\gamma, c)$ is the standard cost function and $\lambda R(\gamma)$ is the regularisation term weighted by λ. $R(\gamma)$ is either L2 or L1 norm and λ is a hyper-parameter which controls the contribution of the regularisation term to the standard cost function.

2.6.2 Dropout

Dropout is a powerful technique used when regularising bigger networks introduced by Srivastava et al. (2014). For each training example, some neurons are randomly removed from the network as well as their incoming and outgoing connections via a certain probability p. Dropout is viewed as sampling and evaluating reduced networks from the whole network at training time. Each training example is trained on a diminished network whose neurons survived dropout. All reduced networks share weights extensively.

At test time, it is practically impossible to average predictions from all reduced networks. However, there is a way to use a single model that approximates the resulting model from all reduced models without using dropout at test time. The core idea is to scale down by multiplying by p at test time all the weights from a neuron trained with a probability p. This guarantees that, at a given layer, the output under dropout and the actual value at test time are the same. By means of this scaling process, all reduced models are combined into a single one referred to as an approximate averaging model at test time.

DNNs generalise well when they are trained on enough data. However, sometimes the amount of data is limited. Data augmentation technique can be used to generate new images from the original ones to augment the dataset size and avoid over-fitting issue.

2.6.3 Data augmentation

Deep learning models are known to require large amount of data for good performance. They involve a large number of parameters which in turn requires more data to get adjusted. More data helps diagnose the over-fitting issue, but data is not always available. The most commonly used technique in computer vision is data augmentation. Data augmentation is a strategy used to generated more images referred to as artificial images from the original data. This technique is mostly used in computer vision to (a) reduce the over-fitting problem by generating more data, (b) not allow the image to appear exactly the same throughout the training process. This strategy seems to be straightforward for some task like object classification. Operations like translation, rotation, mirroring, colour shifting (Krizhevsky, Sutskever, and Hinton, 2012), random cropping, shearing images can be effective and improve the model performance (Goodfellow, Bengio, and Courville, 2016). However, one has to care about transformations which can change the correct class. Practically, two forms of data augmentation are observed:

1. **Off-line data augmentation**: This kind of data augmentation is performed before feeding images to the model and saved on the disk. It is the best option when

dealing with a small dataset. It allows the increase of the dataset size by a factor relatively equal to the number of transformations performed.

2. **Data augmentation on-the-fly** also know as **online data augmentation**: This option of data augmentation seems to be suitable for relatively bigger dataset. To work around the massive increase in size, the augmentation is done on mini-batches before feeding them into the model during the training.

Over-fitting can also be avoided by stopping the learning process once the model has stopped improving. This is the most used regularising form known as early stopping.

2.6.4 Early stopping

Bigger networks with large number of parameters commonly suffer from over-fitting problems. Their training error decreases gradually over time but their validation error decreases for a while and at some point begins to worsen again. One can get reliable models with low validation error by taking into account the parameters at the point in time with minimised validation error. The optimisation process is performed until the validation error has stopped improving for a while rather than waiting the algorithm to reach the minimum. Each time the validation error decreases, either the parameters or the model with parameters of the current state is saved. The last saved state is considered for the best model.

Early stopping is probably the most commonly used regulariser (Caruana, Lawrence, and Giles, 2001; Prechelt, 1998; Bengio, 2012b; Mahsereci et al., 2017) in deep learning due to its effectiveness and simplicity. It runs the gradient descent at once, there is no need trying many values of parameters and the regularization hyper-parameters.

2.6.5 Batch normalisation

DNNs are hard to train due to the change of inputs distribution associated with each layer. It requires the layers to progressively adapt to the new distribution, since each layer inputs are influenced by parameters of previous layers. This implies that, a little change of parameters in the preceding layers affects all succeeding layers. It slows down the learning process by requiring lower learning rates as well as the careful parameter initialisation. There is a technique called batch normalisation (Ioffe and Szegedy, 2015) or simply batch norm designed to overcome this issue.

When training a model such as a logistic regression, normalising the input data, that is, transforming linearly the input data to have the mean zero and unit variance, can enhance the speed of the learning algorithm (LeCun et al., 1998). It enables an efficient model parameter learning.

DNNs do not only have input data but also activations which are inputs associated with different hidden layers. Normalising each layer inputs makes values more stable, speeds up the learning process and makes it easier on the later layers (Ioffe and Szegedy, 2015). Batch norm ensures that the distribution of the non-linearity inputs remains more stable throughout the network. This is done by fixing the mean and variance of layer inputs. It allows the use of higher learning rates without caring about the divergence issue. Given the intermediate value $s^{[h](m)}$ related to the m^{th} training example at layer h, batch norm is implemented as follows:

$$\hat{s}^{[h](m)} = \frac{s^{[h](m)} - \mu}{\sqrt{\sigma^2 + \epsilon}}, \tag{2.61}$$

where $\mu = \frac{1}{l}\sum_{m=1}^{l}\left(s^{[h](l)}\right)$ is the mean over the mini-batch (mini-batches are small batches whose number of examples is less than the total number of training examples. They are used to compute the error and update the model parameters) of size l. Its corresponding variance is yielded by $\sigma^2 = \frac{1}{l}\sum_{m=1}^{l}\left(s^{[h](l)} - \mu\right)^2$. ϵ is a small value added for numerical stability to avoid dividing by zero in case σ^2 would be zero.

Normalising the input of each layer can result in changing what a layer can represent (Ioffe and Szegedy, 2015). To ensure that the transformation introduced in the network is an identity transformation, a pair of parameters $\lambda^{[h]}$, $\eta^{[h]}$ is introduced to scale and shift the normalised value for each activation.

$$\bar{s}^{[h](m)} = \lambda^{[h]}\hat{s}^{[h](m)} + \eta^{[h]}, \tag{2.62}$$

where $\lambda^{[h]}$ and $\eta^{[h]}$ are parameters to be updated through back propagation. They are used to control the value of $\bar{s}^{[h](m)}$.

By stacking together the l training examples in the mini-batch, the vectorised form of (2.62) is defined as follows:

$$\bar{\boldsymbol{S}}^{[h]} = \lambda^{[h]}\hat{\boldsymbol{S}}^{[h]} + \eta^{[h]}. \tag{2.63}$$

Batch normalisation has a slight regularisation effect since activations are scaled by the mean and variance computed on a mini-batch rather than the entire training set. However, enlarging the size of the mini-batch reduces this effect.

During the training process, batch norm processes the data by one mini-batch at a time. However, at test time one wants to make predictions from the network. Theoretically, the entire training set could be run to estimate μ and σ^2. Practically, what is done is to implement an exponential moving average by keeping track of mean and variance seen on mini-batches at training time and use the latest values to roughly estimate the mean and variance at test time.

With deep learning models, the learning occurs through the process of changing the model parameters over time in the direction which minimises the cost function. The parameters are updated via an optimisation algorithm. A number of optimisers have been suggested in the deep learning literature. Some of them seem to work better for various architectures. However, experimenting with them is still advisable. The next section highlights the most common optimisers. Their key role is to update the model parameters to enable the learning process.

2.7 Optimisation algorithms

Applying machine learning is an iterative process where one tries different methods before picking one that performs better. As a result, the speed with which models can be trained is key. Training deep learning models is quite slow when applied to big data, due to the large numbers of parameters in these models. Having quick and good algorithms can

readily speed up the training process as well as the efficiency. This section highlights the gradient descent and its variants as well as the most common optimisation algorithms which seem to work better for various architectures.

2.7.1 Gradient descent and its variants

Intuitively, the gradient descent can be thought about as a challenge of going downhill following the steepest path. The shortest direction seems to be the one following the steepest slope of the hill. Starting at the initial point, one can feel the slope of the hill below his feet and steps down the direction that feels steepest. This process is iteratively repeated until one reaches downhill where the slope is zero or closer to zero.

The slope of a function is referred to as the instantaneous rate of change of the function at a given point. Mathematically, this is the derivative of the function at that point. The gradient is the generalisation of a slope when a function takes on a vector of slopes in different directions. To minimise the cost function, its gradients are computed through back propagation and the parameters are iteratively updated in the negative direction of the gradients.

Batch gradient descent processes the whole training set before making each small step. However, one can get a faster learning algorithm by letting the gradient descent start making progress before finishing to process the entire training examples. This can be done by updating the gradients after each mini-batch.

When the mini-batch size is equal to the size of the entire training set, it ends up being a batch gradient descent. When it is one, each training example is a mini-batch at its own, it leads to another algorithm called stochastic gradient descent. The difference between batch gradient descent, mini-batch gradient descent, and stochastic gradient descent is in the number of training examples used to perform one update step.

Practically, using batch gradient descent with large amount of data is too slow since it updates parameters after processing the entire training set. The stochastic gradient descent loses the vectorisation property since only one training example is processed at time. As a result, the mini-batch gradient descent is practically preferred. It encompasses vectorised implementations and updates parameters progressively. Due to the layout and access of the computer memory, computations are usually fast if the mini-batch size is the power of 2. Usual mini-batch sizes are between 32 and 512. If n is the current epoch, the parameters for the next epoch are updated as follows:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha \, \nabla \boldsymbol{\theta}, \tag{2.64}$$

where α is the learning rate, $\boldsymbol{\theta}$ are the model parametesr either the weights γ or the biases c, and $\nabla \boldsymbol{\theta} = \frac{\partial E}{\partial \boldsymbol{\theta}}$ is the gradient of the cost function with respect to the model parameters.

In deep learning literature, there are other optimisation algorithms which seem to perform better than the mini-batch gradient descent. They use an algorithm called exponentially weighted moving average.

2.7.2 Gradient descent with momentum or momentum

The standard gradient descent takes a lot of oscillations which slow down the convergence process to the minimum. Using a larger learning rate may even end up overshooting and diverging from the minimum. This prevents the use of a larger learning rate. The gradient descent with momentum (Sutskever et al., 2013; Qian, 1999) seems to work faster than the usual gradient descent. It allows the gradients to move towards the minimum rather than oscillate all along. It also allows the use of larger learning rate to speed up the learning process. The core idea is to store the exponentially moving average of the gradients computed on preceding mini-batches in the variable V and use them to update parameters on current mini-batch rather than the actual gradients. V can be thought of as a velocity of a ball rolling downhill, building up speed (and momentum) according to the direction of the gradient of the hill. Implementations are defined as follows

$$V_n = \beta V_{n-1} + (1-\beta)\nabla\boldsymbol{\theta}, \tag{2.65}$$

where V_n is the moving average associated with the gradients $d\boldsymbol{\theta}$, and β is the parameter that controls the exponentially moving average. It is referred to as a kind of velocity that pushes the gradients in the right direction. If n is the current epoch, the parameters corresponding to the next epoch are updated as follows:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha V_n, \tag{2.66}$$

where α is the learning rate.

2.7.3 RMSProp

Another algorithm which can speed up the gradient descent is Root Mean Squared Prop (RMSprob) introduced by Tieleman and Hinton. It aims at damping off the oscillations which slow down the gradients. It speeds up the gradients in the direction towards the minimum and enables the use of larger learning rate. RMSProp stores the exponentially moving average of the gradient squared computed on previous mini-batches and use them to update parameters on the current mini-batch.

$$W_n = \beta W_{n-1} + (1-\beta)\nabla^2\boldsymbol{\theta}. \tag{2.67}$$

The parameter β plays the same role of speeding up the gradients in the direction towards the minimum. W_n is the exponentially moving average associated with the past gradients squared.

The parameters are updated as follow:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \frac{\alpha}{\sqrt{W_n}+\epsilon}\nabla\boldsymbol{\theta}, \tag{2.68}$$

where α is the learning rate. For numerical stability an ϵ term is added to the denominator to avoid dividing by zero in case W_n should be zero. Another algorithm which seems to work even better in most of the cases in computer vision is Adam optimisation algorithm.

2.7.4 Adam optimisation algorithm

Adam optimisation algorithm (Kingma and Ba, 2014; Anonymous, 2018) combines both the momentum and RMSProp properties. In addition to storing the exponentially moving average of the gradients and their squared, it computes the bias correction associated with them and use the results to update parameters on the current mini-batch. The name Adam is derived from adaptive moment estimation. The bias corrected version is given by

$$\hat{V}_n = \frac{V_n}{1-\beta_1^n}, \tag{2.69}$$

$$\hat{W}_n = \frac{W_n}{1-\beta_2^n}, \tag{2.70}$$

where V_n and W_n are respectively the bias correction related to the gradients and gradients squared.

The parameters are updated as follows:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \frac{\alpha}{\sqrt{W_n}+\epsilon}\hat{V}_n, \tag{2.71}$$

where α is the learning rate hyper-parameter. The authors of the Adam algorithm proposed the values of β_1 and β_2 to be 0.9 and 0.999 respectively, and 10^{-8} for ϵ (Kingma and Ba, 2014).

In machine learning literature, a number of model evaluation metrics are encountered. The next section mentions some metrics used to assess the classification and regression models. They will be invoked when evaluating the application models in chapter 3 and 4.

2.8 Model assessment

2.8.1 Model evaluation procedures

When training a model, the interest is not in the performance on the training dataset but in generalisation on unobserved samples. Training and testing a model on the same dataset results in complex model which over-fits the data and does not necessary generalise on unseen dataset. In the practice, two common evaluation procedures are encountered – training/validation split and k-fold validation (Fushiki, 2011)

1. Splitting the dataset into training and validation/test datasets: To get the sense of how the model is gaining the capability of generalisation, the model is trained on training dataset but evaluated on a different dataset. This procedure yields a good estimate of holdout sample performance. This procedure is handy and commonly used due to its simplicity, speed, and flexibility.

2. K-fold cross-validation: This procedure splits the dataset into K train/validation subsets. Each time the model is trained on K-1 subsets and validated on the remaining one. This procedure mostly results in good estimate of holdout sample

performance. However, its computational time is K times slower than the train/test split strategy.

This thesis adopts the train and validation/test split approach since deep learning models are computationally intensive. This section highlights some commonly used metrics for classification and regression models. They will be used to assess the performance of the models fitted in the applications chapters.

2.8.2 Metrics for classification models

Let $D, t(i)$, and $\hat{t}(i)$ be respectively a set of instances, the observed and predicted value associated with an instance i where $t(i)$ and $\hat{t}(i)$ are binary, either 0 (negative class) or 1 (positive class).

1. The accuracy gives the proportion of correctly predicted instances over the total number of instances.

$$\text{accuracy} = \frac{1}{|D|} \sum_{i \in D} I\left[\hat{t}(i) = t(i)\right], \tag{2.72}$$

   where $I[\cdot]$ is and indicator function which is 1 if the expression $[\cdot]$ holds, otherwise 0.

2. The sensitivity gives the probability that a positive instance i will be correctly classified.

$$\text{sensitivity/recall} = \frac{\sum_{i \in D} I\left[\hat{t}(i) = t(i) = 1\right]}{\sum_{i \in D} I\left[t(i) = 1\right]} \tag{2.73}$$

3. The specificity gives the probability that a negative instance i will be correctly classified.

$$\text{specificity} = \frac{\sum_{i \in D} I\left[\hat{t}(i) = t(i) = 0\right]}{\sum_{i \in D} I\left[t(i) = 0\right]}. \tag{2.74}$$

4. The precision is the proportion of the number of positive instances correctly predicted over the number of instances predicted positive.

$$\text{precision} = \frac{\sum_{i \in D} I\left[\hat{t}(i) = t(i) = 1\right]}{\sum_{i \in D} I\left[\hat{t}(i) = 1\right]} \tag{2.75}$$

5. $F1$ represents the weighted average of precision and recall.

$$\text{F1 score} = 2 \times (\text{recall} \times \text{precision}) / (\text{recall} + \text{precision}). \tag{2.76}$$

6. The area under the receiver operating characteristic (ROC) curve shortly denoted as AUC (Bradley, 1997; Provost et al., 1998) represents the discriminative ability of correctly distinguishing between classes. It can be viewed as the proportion of the unity squared area, its value always lies between zero and one. The perfect model

would have an AUC of 1 and the worst an AUC of 0. If a model does not achieve an AUC greater than 0.5, it performs no better than random guessing.

2.8.3 Metrics for regression model

In regression models, the sum of the squares total (SST) evaluates the difference between data points and the data mean. The sum of square errors (SSE) assesses the difference between the data points and the model predictions. The difference between SST and SSE gives the model improvement over the mean model prediction (Saunders, Russell, and Crabb, 2012), which always outputs the data mean as prediction.

1. Dividing the distance between SST and SSE gives the coefficient of determination also known as R-squared (R^2). It is a statistical measure used to assess the goodness of the regression model (Chen, 2002; Saunders, Russell, and Crabb, 2012). Its mathematical formulation is given by

$$R^2 = 1 - \frac{SSE}{SST}. \tag{2.77}$$

   R^2 value lies between 0 and 1. A model with $R^2 = 0$ does not perform better than the mean model while $R^2 = 1$ denotes a model with perfect predictions.

2. Dividing the SSE by the number of data points gives the mean squared error (MSE).

The intersection over union (IoU) also called the Jaccard index is the commonly used metric to assess the similarity between two shapes (Rezatofighi et al., 2019). The IoU encodes the shape properties of the objects under comparison into the region property and produces a normalised measure that focusses on their areas. This property makes the IoU invariant to the scale of the problem under consideration. Due to this important feature, all performance measures used to evaluate computer vision tasks such as detection (Lin et al., 2014), segmentation (Cordts et al., 2016; Zhou et al., 2017), and tracking (Kristan et al., 2017) rely on this metric. If A and B are shapes, the IoU is defined as follows:

$$IoU = \frac{\text{Area of overlap}}{\text{Area of union}} = \frac{A \cap B}{A \cup B} \tag{2.78}$$

Chapter 3

Automated detection of bounding boxes in images

The main objective of this thesis is to build a machine learning algorithm to automate the HBW and WLT individual identification task based on the shape of HBW fin and special marks, and unique spot on the WLT back. However, for some images, the salient or informative object (toad or whale) only occupies a small space in the image, Figure 3.1 displays example images of this category. This causes the spots on the WLT's back and the shape of HBW fin and special marks to not be clearly visible while they are the key feature for animal individuals. Thus, extracting the entire salient object in the images is crucial for the identification task in this thesis. Though it is possible to crop tens to hundreds of images manually, cropping thousands of images is tiresome, time-consuming, and needs some automation.

When it comes the task of detecting and localising the object of interest in the images, computer vision algorithms rely on CNNs and bounding boxes (Rahman et al., 2019). For image processing, the bounding box is referred to as coordinates of a rectangular border which entirely encloses a digital image on bi-dimensional background. The bounding box is the core component of computer vision object detection algorithms such as Region-based Convolutional Neural Network algorithm (Girshick, 2015; Ren et al., 2015) and You Only Look Once referred to as the YOLO algorithm (Redmon et al., 2016). The inputs to a bounding box model are pairs of images and the bounding box data associated with the salient object in the images. The bounding box data (Girshick et al., 2014) are the coordinates (x, y, w, h), where (x, y) are the coordinates of the upper-left corner of the bounding box, w and h are respectively the width and the height of the salient object in the image. The four numbers corresponding to the bounding box data suffice to define the coordinates associated with the four corners of the bounding box.

The objective of this chapter is to build a convolutional neural network to automate the detection and localisation of the salient object in the images. Localised salient objects will be extracted by cropping images around them. In this thesis, the task of detecting and localising the salient object in images using the bounding box is referred to as bounding box detection. Generating the bounding box data manually is challenging. To solve this task, a semi-automated algorithm which utilises image thresholding and binarization (Chaki, Shaikh, and Saeed, 2014; Gatos, Ntirogiannis, and Pratikakis, 2009) to create the bounding box data is implemented. This algorithm involves a threshold. It is impossible to choose a threshold which can be used to accurately crop all images at once, hence this algorithm is semi-automated. The generated bounding box data are used to train a CNN for bounding box regression. The model outputs four numbers estimating the four numbers defining the bounding box data. The model is trained taking into account some

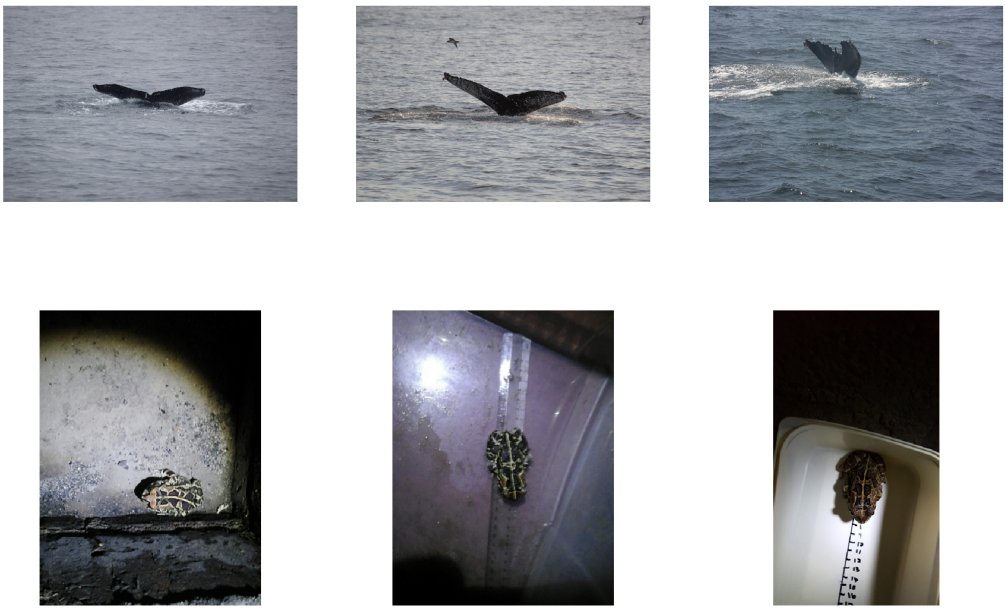


Figure 3.1: Example of images whose salient object only occupies a small space in the image

or a combination of concepts such as data augmentation, data replication, and cyclic learning rate. For the rest, this chapter describes the data, the methods used to generate the bounding box data and train the model, and concludes with the presentation and discussion of the model results.

3.1 Data

3.1.1 HBW dataset

The data consists of 25,361 images originally collected by different institutions across the globe and uploaded to the Happywhale platform with the aim of identifying individual marine mammals. Some of the images are coloured (RGB) and others grayscale, and differ in terms of size and resolution. All the images are taken at the sea, their backgrounds do not differ too much.

3.1.2 WLT dataset

This dataset comprises of almost two thousand images collected by citizen scientists in South Africa. They were either uploaded to iSpot , a citizen science project that collects images or sent to the WLT project, a conservation project staffed by volunteers. All the images are coloured but different in terms of size and resolution. Images of this dataset

have different background. The majority has a white background. The following group of images have a black background. The remaining are taken in the gardens, inside the box, and elsewhere.

3.2 Methods

3.2.1 Method used to obtain bounding boxes for training data

In some images, the salient objects only occupy a small space. Cropping them not only makes the key features which are spots on the WLT's back and the shape of HBW fin and special marks more visible but also speeds up computations by limiting the processing to the salient objects rather than the whole images.

(a) Semi-automated algorithm:

Cropping images usually involves edge/outline detection. The pixel intensity in the image ranges from 0 to 255. A pixel with 0 intensity is black and the one with 255 is white. As a result, pixels whose intensities are closer to 0 tend to be black and the ones closer to 255 white. The main target here is to extract the salient object from the background. This is done by assigning to the background pixels of the same intensity. If the background is black, the salient object will be white or vice versa. The semi-automated algorithm is implemented as follows:

1. The image is converted into the grayscale mode.
2. The grayscaled image is then thresholded and binarized (Chaki, Shaikh, and Saeed, 2014; Gatos, Ntirogiannis, and Pratikakis, 2009) by assigning an intensity of 255 to pixels whose intensity is beyond the threshold and 0 for pixel intensity beneath the threshold.

After image binarization, one of the following three scenarios is observed.

1. If the threshold set is the correct one, the outlines of the background and the object of interest will be separated, one will be black and the other white, as shown by BINARIZED IMAGE in Figure 3.2. The salient object is assumed to be bigger than other objects in the image. Its corresponding outline is extracted by considering the longer outline among the outlines of all outlined objects in the image. The extreme points: the leftmost, rightmost, topmost, and bottommost along the outline of the salient object are used to set the bounding box as shown by EXTREME POINTS and BOUNDING BOX images in Figure 3.2.
2. If a part of the object of interest has the same colour as the background, this part will be cut out. For this case, setting a bounding box around the object of interest separated from the background results in extracting only a piece of the salient object.
3. If the background pixel intensity is the same as the salient object, the algorithm completely fails to separate the salient object outline from the background. As a result, the algorithm returns the outline of the whole image as the bounding box. This case is likely observed when the background and the salient object look alike.

(b) Generating the HBWs and WLTs bounding box data:

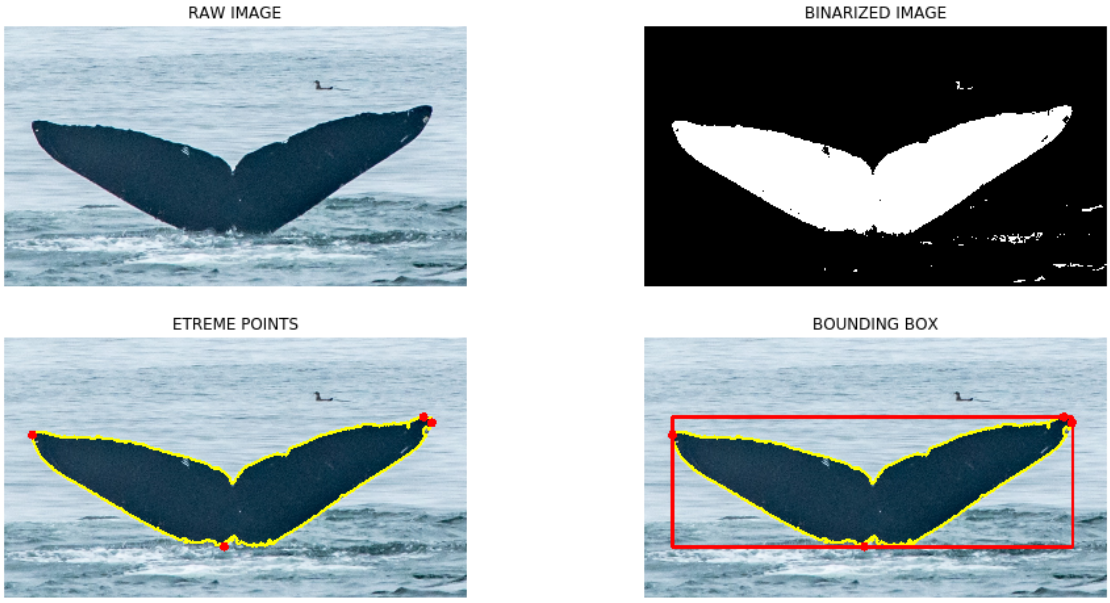


Figure 3.2: RAW IMAGE is the original image before any manipulation. BINARIZED IMAGE is obtained after thresholding and turning pixels to be either white or black. EXTREME POINTS are the topmost, bottommost, leftmost, and rightmost points along the outline of the salient object in the image. BOUNDING BOX is a rectangle localising the salient object in the image defined using extreme points.

The application of the semi-automated algorithm to the HBWs and WLTs results in the three scenarios discussed above. For each threshold set manually by the user, some images are accurately cropped and others are not. The ones that are accurately cropped can be referred to as easy examples and the remaining hard examples for this threshold. However, some images can be easy for one threshold and hard for another or vice versa. It is not possible to choose a single threshold which can be used to accurately crop all images at once. Hence, this algorithm is semi-automated rather than fully automated. The bounding box data (Girshick et al., 2014) is defined by the four numbers (x, y, w, h), where (x, y) are the coordinates of the upper-left corner of the bounding box, w and h are respectively the width and the height of the animal in the image. Starting with the upper-left corner of the bounding box and following the clockwise direction, the coordinates associated with the four corners can be defined as follows: $(x, y), (x + w, y), (x + w, y + h)$ and $(x, y + h)$.

1. **HBWs bounding box data**: For the HBW dataset, a threshold of 120 was used for all images. Images were saved with a bounding box set around the HBW tail in the image for the sake of manual selection. 4,962 images whose HBW tails were accurately located by the bounding box were selected and divided into three categories, 4,000 images for training, 700 images for validation and 262 images for testing.

2. **WLTs bounding box data**: The WLT dataset was relatively small. To obtain up to a thousand of bounding box images, three different thresholds were used. Each time, images were saved with a bounding box set around the WLT in the image to facilitate the manual check. With a threshold of 140 the algorithm accurately

cropped 595 images. Then 320 images were selected with a threshold of 100. Finally, a threshold of 70 was used and the algorithm cropped 283 images. It is worth noting that each time the algorithm was applied to the remaining images. In total, 1,195 images were selected and 1,000 images were used for training, 131 for validation, and the remaining for testing.

It is noteworthy mentioning the following:

(a) For both HBW and WLT datasets, only a subset of the whole dataset was cropped. The selected images are used to train a CNN which will be used to crop the remaining images including the hard images which were not accurately cropped by the semi-automated algorithm.

(b) Besides the training, validation, and test datasets, two samples of 53 HBW and 28 WLT hard example images which were not accurately cropped by the semi-automated algorithm are selected. They bounding boxes are manually generated. These samples are used to assess how the model performs on hard example images.

(c) For WLT dataset, a slightly high threshold was suitable for images with a white background and a slightly low threshold was convenient for images with a black background. For the HBW dataset, since all images were taken at the sea, the background was not that much different for many images. As a result, with a single threshold, the algorithm managed to accurately crop many images at once compared to WLT dataset.

3.2.2 Method used to train the CNN for bounding box regression

(a) Model architecture:

The model used a CNN employing 6 convolutional blocks followed by two fully connected layers. The inspiration behind the architecture construction was drawn from VGG16 (Simonyan and Zisserman, 2014b) architecture. Each convolutional block is a stack of 3 convolutional layers. Each convolutional layer utilises 64 filters with a kernel size of 3. Down-sampling was directly performed via a 2×2 convolution with stride 2. This strategy has been used by He et al. (2016a). The batch normalisation and dropout were applied after each convolutional block. These two techniques are reported to avoid the overfitting issue (Ioffe and Szegedy, 2015; Srivastava et al., 2014). Intuitively, to be able to capture the coordinates associated with the top-left, top-right, bottom-left and bottom-right corners of the bounding box, after convolutional blocks, max pooling was performed separately on rows and columns and the results were concatenated. The ReLU non-linear activation (Nair and Hinton, 2010) was used across hidden layers. This activation function only activates positive features. It converts negative features to zero, which makes the network sparse and easy to train efficiently. The fully connected layers had 16 nodes and 4 nodes respectively. The output layer used a linear activation function and 4 nodes as the outputs of the model are 4 numbers estimating 4 numbers defining the bounding box data. Several architectures were tried consisting of 4, 7 or 8 convolutional blocks, using 64-256 filters with the kernel size between 3 and 5, with and without dropout. The final architecture was selected based on the lowest validation mean squared error (MSE) loss.

(b) Hyper-parameters:

The model was trained using the cyclic learning rate. According to Smith (2015), this strategy eliminates the need for tuning the learning rate. Rather than using a fixed learning rate, this technique fixes the upper and lower bounds and cycles the learning

rate between them a number of times referred to as the number of cycles. After a cycle length which is the number of iterations (epochs) allocated to each cycle, the model continues to train with its current weights, but it resets the value of the learning rate to the maximum to start off a new cycle. The model repeats the same scenario until it finishes the number of cycles. For this application, the maximum learning rate was 0.032 and the minimum 0.002. The learning rate was reduced by a factor of 0.25 each time no improvement is made over 3 epochs. When the learning rate is reduced down to the minimum, the model computed the remaining epochs of the current cycle using the minimum learning rate. The number of cycles was 3 with a cycle length of 100 epochs for each one. The training was stopped by means of early stopping if no improvement is made over 10 epochs. The dropout rate was set to 0.2 in the convolutional layers and 0.5 in the fully connected layers. The weights were initialised according to He et al. (2015) initialisation rule. The Adam (Kingma and Ba, 2014) optimising algorithm was used to update the model parameters. An experimentation with different values for the dropout rate between 0.1 and 0.5 was carried out. Various bounds for the cyclic learning rate between 0.0001 and 0.1 were tried. Some of these values led to higher training MSE loss and others to lower training MSE loss but higher validation MSE loss. Different optimisation algorithms like SGD with and without momentum (Sutskever et al., 2013) and RMSprop ("RMSprop Gradient Optimization") were also tried. The final values were chosen on the basis of lower validation MSE loss.

(b) Variance normalisation:

Batch normalisation (Ioffe and Szegedy, 2015) and dropout (Srivastava et al., 2014) are both powerful concepts when training deep learning models. However, using both concepts at the same time needs some attention. Performing batch normalisation after dropout slightly reduces the model performance (Li et al., 2018). During training, dropout randomly zeros some results but rescales up the rest to keep the overall average. As a result, it does not preserve the output variance during training and when undertaking inference (Li et al., 2018)

The behaviour of batch normalisation also differs from training and inference phases. At training time, batch normalisation is carried out on mini-batches. Simultaneously, the moving average of the mean and variance are computed. During inference, the moving average is referred to as the sample estimation. Nevertheless, the moving average is not the right estimate of the variance since dropout behaviour has altered the variance (Li et al., 2018).

As a workaround, one of the solutions suggested is recomputing the moving average of the mean and variance without dropout by maintaining other layers fixed. Only layers comprising of batch normalisation are re-run. The performance is expected to be slightly better.

3.2.3 Data manipulations

(a) Data replication and augmentation:

Data augmentation was used to increase the number of images seen by the model during training. This has been shown to reduce over-fitting (Mikołajczyk and Grochowski, 2018; Taylor and Nitschke, 2017). Images were augmented via an affine transformation (Perez and Wang, 2017) which composed rotation, shearing, zooming, and shifting. The rotation range was between -5 and +5 degrees, the shear range was between -5 and +5, the image height or width was randomly zoomed within a range of 0.9 and 1.0. Either the image

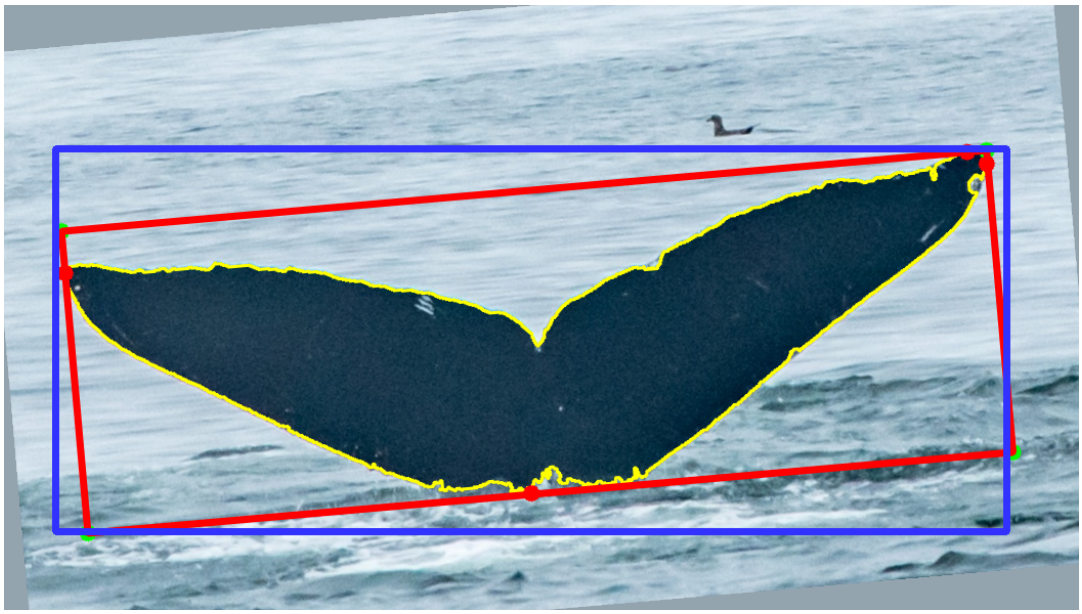


Figure 3.3: The red bounding box is the original bounding box. The blue is the recomputed bounding box once the image has been rotated by 10 degrees anti-clockwise.

height or width was shifted by a factor of 0.01× height or width. Some informal trial-and-error experimentation was used to select these parameters. However, the rotation and shear range were kept relatively small to avoid getting a HBW tail position which is unlikely to occur since in almost HBW images the HBW tail is in the horizontal position. It is worth mentioning that the same values were used for both datasets.

(b) Images resizing and normalisation:

Images were resized via an affine transformation which transformed a rectangular area of the original images to a square area of 128 × 128 pixels. Images were centred, compressed to the mean ratio and grayscaled. The transformed image was annotated with the transformed bounding box. In Figure 3.3, the red line shows the original bounding box whilst the blue is the recomputed bounding box after a rotation of 5 degrees anti-clockwise. Image pixels were normalised to zero mean and unit variance to speed up the learning process (LeCun et al., 1998). During the validation phase, the data augmentation was skipped since the validation set is required to reflect the real data. Images are only compressed to the mean ratio, resized, normalised to zero mean and unit variance, and grayscaled. The left image in Figure 3.4 was transformed for the validation phase and the right for the training phase.

3.2.4 Models

This section describes models fitted for both HBW and WLT datasets.

(a) HBW models:

Three models were trained using the same architecture, the same 4,000 original images and they were validated over 700 images.

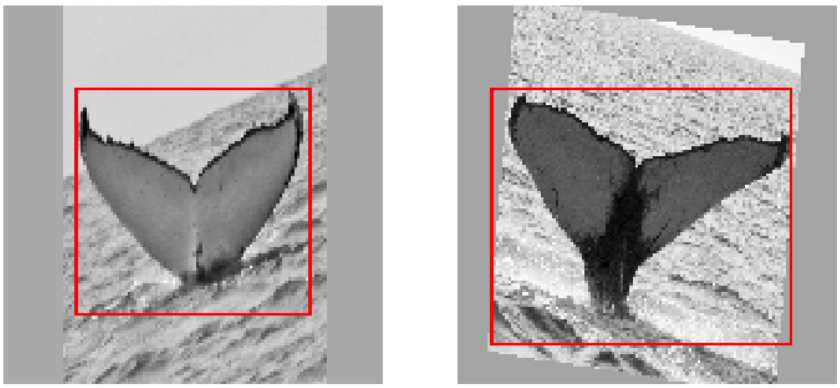


Figure 3.4: Both images are grayscaled, compressed to the mean ratio, resized, normalised and annotated with the transformed bounding box via an affine transformation. Additionally, the right image is randomly transformed through online data augmentation. The right image is utilised during training and the left during the validation phase.

1. The first model referred to as **W1** in Table 3.1 utilised 4 augmented versions of each original image, with data augmentation done online (Section 2.6.3). As a result, this model augmented four times the size of the training dataset.
2. The second model named **W2** in Table 3.1 applied online data augmentation to transform the original images so that an image does not appear the same throughout the training process. However, it did not augment the size of the training dataset.
3. The last model referred to as **W3**, was solely trained on the original images. Images were not transformed, they appeared the same throughout the entire training process.

(b) WLT models:

Two models were fitted using 1,000 original images and validated them using 131 images. Each model utilised 16 augmented versions of each image, with data augmentation performed online. Their results are summarised in Table 3.2.

1. The first model referred to as **T1** in Table 3.2 was trained from scratch by randomly initialising weights and optimising them through an iterative process during training.
2. The last model named **T2** in Table 3.2 applied the transfer learning technique (Tan et al., 2018). This case experimented if the model can utilise the knowledge gained from localising the HBW tail in the photograph, to localise the WLT in the photograph. This was motivated by the fact that CNNs learn more generic features in the earlier layers and deal with complex objects in later layers (Bengio, 2012a). As a result, to be able to leverage the low-level and mid-level features learned from HBWs and generalise on WLTs, the weights in all 4 convolutional layers are initialised to the values obtained using the best HBWs model. Weights in the first convolutional layers are not estimated, they are held fixed at these values. Weights in the final two convolutional layers are estimated. In other words, they fine-tuned from these initial values.

3.2.5 Model assessment

The models were assessed by the means of the mean squared error (MSE) loss, the coefficient of determination R-squared (R^2), and the intersection over union (see section 2.9.3). The MSE was computed between original bounding box data values $(x_i, y_i, w_i, h_i)_{i=1}^N$ and estimated values $(\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i)_{i=1}^N$, where N is the number of validation images. For this case, R^2 gives the proportion of model prediction improvement compared to the mean model which approximates each validation bounding box data values using the mean values of the validation dataset $(\bar{x}, \bar{y}, \bar{w}, \bar{h})$, where $\bar{x} = \frac{1}{N}\sum_{i=1}^N x_i$, etc. The intersection over union gives the ratio of the overlapping area of the ground truth bounding box and the predicted bounding box to the total area. The IoU of two perfectly overlapped bounding boxes is 1 and 0 if there is no overlap.

3.2.6 Software, libraries and computational resources

Data augmentation and models were implemented using the Python programming language and the deep learning framework Keras. The Python data analysis library Pandas was used to read and manipulate .csv files. Images were loaded and visualised using the Python Image Library (PIL). An affine transformation function from SciPy, the Python scientific computing library was used through the transformation of images. Some metrics from the scikit-learn library were used to assess the model performance. The South African Centre of Higher Performance Computing CHPC platform also known as Lengau cluster was utilised for computational resources. A memory storage of 125 GB was available and each Python script implementation was submitted as a job in a queue and the scheduler assigned it to a compute node of 24 cores, 24 CPUs, and 128 GB of memory. The semi-automated algorithm was implemented locally using Python programming language and Open Computer Vision library (OpenCV).

3.3 Results

3.3.1 HBW model results

Table 3.1 summarises the ability of various neural network models to detect bounding boxes in the HBW validation dataset. Models are assessed using the MSE and the coefficient of determination $\mathbf{R}^2$ and the intersection over union **IoU**. Models are trained using the cyclic learning rate strategy. Some use online data augmentation and replication to transform and augment the training images. The best model **W1** achieved the intersection over union **IoU** = 0.91 and $\mathbf{R}^2$=0.94 over a test set consisting of 262 images. Figure 3.5 displays the bounding box detection ability of the best model, on validation images and hard examples images drawn from images which were not accurately cropped by the semi-automated algorithm. The same best model achieved a coefficient of determination $\mathbf{R}^2$ = 0.852 on a sample of 53 HBW hard example images which were not accurately cropped by the semi-automated algorithm.

3.3.2 WLT model results

Table 3.2 summarises the ability of two neural network models to detect bounding boxes in WLT validation images. Models are assessed using the **MSE**, $\mathbf{R}^2$, and **IoU**. They are

Model	Ag	Re	C1	C2	C3	ABN	R^2	IoU
W1	Yes	Yes	19.01	15.42	14.63	**14.26**	0.914	0.90
W2	Yes	No	37.37	19.49	17.41	**16.16**	0.903	0.88
W3	No	No	27.29	25.57	22.89	**21.72**	0.897	0.87

Table 3.1: Assessment of the ability of various neural networks to detect bounding boxes on the humpback whales validation dataset. Models are assessed using the **MSE** and the coefficient of determination $\mathbf{R}^2$. Each model was trained using the cyclic learning rate with 3 cycles of a cycle length of 100 epochs each. **C1,2,3** columns represent respectively the validation mean squared error (**MSE**) after the first, second, and third cycle. **ABN** column denotes the **MSE** after recomputing the batch normalisation. The last column represents the goodness of the model assessed by the $\mathbf{R}^2$ criterion. For each model, $\mathbf{R}^2$ is computed for the case of the lowest **MSE** (bold number). **Ag** notifies if the model uses online data augmentation to transform training images and **Re** if the model uses augmented versions of the original images to increase the training dataset size. **IoU** gives the intersection over union of ground-truth and predicted bounding box areas

trained using the cyclic learning rate strategy and use data augmentation and replication to transform the training images and augment the size of the training dataset. The best model (**T2**) achieved the **IoU** = 0.87 and $\mathbf{R}^2$= 0.85 over a test set consisting of 64 images. Figure 3.6 shows the bounding box detection ability of the best model, on WLT validation images and hard images taken from images which were not correctly cropped by the semi-automated algorithm. The best model achieved the coefficient of determination $\mathbf{R}^2$=0.803 on a sample of 28 WLT hard example images which were not correctly cropped by the semi-automated algorithm.

Model	Ag	Re	C1	C2	C3	ABN	R^2	IoU
T1	Yes	Yes	31.40	31.32	**30.41**	32.66	0.823	0.84
T2	Yes	Yes	26.71	30.23	27.92	**26.56**	0.846	0.86

Table 3.2: Assessment of the ability of two neural networks models to detect bounding boxes on the WLT validation dataset. Models are assessed using the **MSE**, $\mathbf{R}^2$, and **IoU**. Each model was trained using the cyclic learning rate with 3 cycles of length of 100 epochs each. **C1,2,3** columns represent respectively the **MSE** after the first, second, and third cycle. **ABN** column denotes the **MSE** after recomputing the batch normalisation. The last column represents the goodness of the model assessed by the $\mathbf{R}^2$ criterion a. $\mathbf{R}^2$ is computed only for the case associated with the lowest **MSE** (bold numbers). **Ag** and **Re** indicate if the model used data augmentation and augmented versions of the original images. **IoU** is the intersection over union of ground-truth and predicted bounding box areas.

3.4 Discussion of the model results

The coefficient of determination and IoU criteria showed that the best models on HBW and WLT datasets achieved reliable performance. The coefficient of determination $\mathbf{R}^2$ and **IoU** of the best model were respectively 0.94 and 0.91 for HBW test set and 0.85 and 0.87

for WLT test set . As shown in Figure 3.5 and Figure 3.6, the neural network models have been able to reliably estimate the bounding boxes in images for which the semi-automated algorithm failed to accurately detect and locate the animal in the photograph.

For HBW dataset, the model **W1** which utilised online data augmentation and replication outperformed the remaining models. The model **W2** which only used online data augmentation outperformed the last model **W3** which did not apply data augmentation. For the 3 cycles performed, the lowest **MSE** is observed after the third cycle. Recomputing batch normalisation without dropout by maintaining other layers fixed slightly lowered the **MSE** for all models. For this dataset, data augmentation, replication, recomputing the batch normalisation, and cycling the learning rate for a number of cycles slightly contributed to the overall performance of the model.

For the WLT dataset, the fine-tuned model **T2** outperformed the model **T1** trained from randomly initialised weights. Recomputing the batch normalisation for the model **T1** resulted in slightly higher **MSE**. For the fine-tuned model, the first cycle achieved the smallest MSE among the 3 cycles. The same scenario was observed throughout the trials performed.

It can be observed that models fitted on HBW dataset achieved slightly higher performance in terms of **MSE** and coefficient of determination $\mathbf{R}^2$ compared to models trained on WLTs. It might be due to the fact that models trained on HBWs used 4,000 original images while ones fitted on western leopard toads only used 1,000 original images. As a result, improving further the results of these models would require more data. Cycling the learning rate for a couple of cycles was helpful for models trained from scratch but not for a fine-tuned model. This seems intuitive since training a DNN from random weights requires optimising a large amount of parameters through an iterative process. As a result, the model converges after many iterations. In contrast, a fine-tuned model keeps a large amount of the parameters fixed and uses them to estimate the remaining ones.

Model usage:

Using the model involves the following 4 steps:

1. Image preprocessing phase: An image passes through an affine transformation that compresses it to the mean ratio, resizes, normalises, and grayscales it.
2. Bounding box estimation: The model is used to estimate the bounding box on the transformed image
3. Bounding box in the original image: The inverse of the affine transformation used to transform the image is used to get the bounding box in the original image.
4. Extraction of the salient object: The salient object inside the bounding box is cropped and saved as the new image one is interested in.

This chapter has shown that neural network models can reliably detect and locate animals in the photographs. To save time and effort, the selected models are used to crop the remaining images and take them to the next step of image matching in the next chapter.

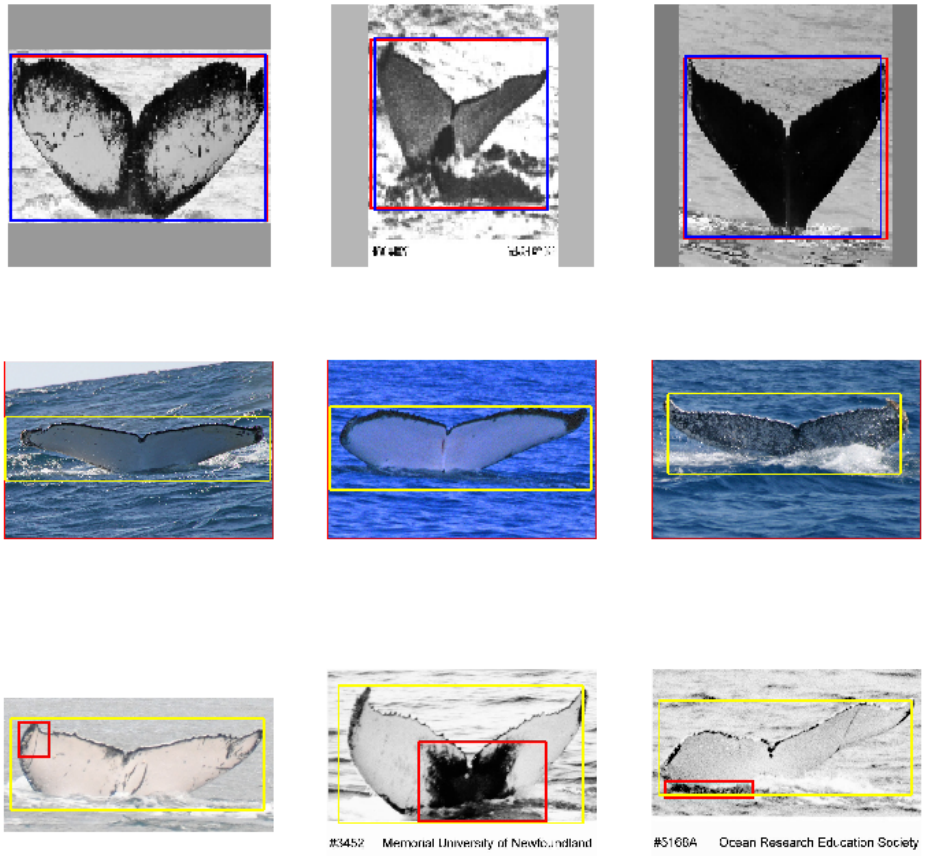


Figure 3.5: Images of the first row are drawn from HBW validation dataset. The red is the original bounding box and the blue is estimated by the best model on HBW dataset. Images of the last two rows are taken from hard examples which were not accurately cropped by the semi-automated algorithm. The second row refers to images for which the semi-automated algorithm completely failed to detect the animal in the image and returned the bounding box of the entire image. The last row concerns images for which the semi-automated algorithm only cropped a small part of the salient object. In the last two rows, the red is the bounding box set by the semi-automated algorithm and the yellow is estimated by the model.

Figure 3.6: Images of the first row are taken from WLT validation dataset. The red is the original bounding box and the blue is estimated by the best model. Images of the last two rows are taken from hard images which were not accurately cropped by the semi-automated algorithm. The yellow bounding boxes are estimated by the best model.

Chapter 4

Identification of individual animals from images with a siamese neural network

4.1 Introduction and study objectives

Individual identification is one of the key tasks performed by ecologists. However, many of the ecologists are still performing this task manually . For example citizens were used to categorise 1.2 million camera trap images from Serengeti National Park (Swanson et al., 2015). Like fingerprints in human beings, some animals in nature are dotted with individual specific features which set them apart from one another and hence can enable their identification (Gomez et al., 2016; Beijbom et al., 2016; Körschens, Barz, and Denzler, 2018). WLTs and HBWs are part of the species dotted with individual-specific markings. Each WLT individual can be identified by the unique spots on their back. Each HBW individual can be identified by the shape of their tail fin and special marks. Though one might be able to manually identify a few individuals from images, processing thousands of images produced annually is tedious, time consuming and leaves out a huge amount of data unexploited and underutilised. Hence, to help ecologists automate the HBW and WLT individual identification, this chapter aims at:

1. Implementing a machine learning algorithm to automate the individual identification task. This is achieved through individual photo matching based on individual-specific marks and extracting individual IDs from obtained matches (images of the same individual). Use the algorithm to individually identify HBWs and WLTs based on their body-marks.
2. Using the implemented algorithm to test the effectiveness of the semi-supervised approach on WLTs.

For the remaining, this chapter describes the data, methods used to train and assess the models, the results from the models and the semi-supervised learning process, and concludes with a discussion of the results.

4.2 Data

4.2.1 HBW dataset

The data was originally collected by different research institutions across the globe and uploaded to the Happywhale platform. It consists of 25,631 images collected from 5,005 unique HBW individuals. The majority of the HBWs has one or two images. 228 individuals have more than 10 images and the individual with most images has 73 images. However, among them there is a HBW name referred to as **new-whale** which comprises 9,664 images. They are individuals appearing only once grouped together under the **new-whale** name. The model architecture which will be used to do individual photo matching needs training on both matched pairs (two images from the same individual) and unmatched pairs (two images from different individuals). This requires individuals with at least two images so that each image can appear in both matched and unmatched pairs during training. After removing individuals with a single image, individuals under the **new-whale** name, and images deemed ambiguous, the HBW dataset remained with 2,915 individuals comprising 13,478 images. Figure 4.1 displays 10 image examples drawn from ambiguous images. This category includes images whose fluke is not visible, two different HBWs in the same image, images whose underside is not visible or only a small part of the image is visible, and images whose fluke points downwards. Figure 4.2 displays the distribution of HBW images from individuals with at least two images.

4.2.2 WLT dataset

This dataset consists of almost two thousand images collected by citizen scientists in South Africa. They were either uploaded to iSpot, a citizen science project which collects images or sent to the WLT project, a conservation project staffed by volunteers. This dataset is much smaller and thus a much harder problem. It is partially labelled. The labelled part has 164 unique individuals comprising at least two images each, the individual with most images has 7 images, and all individuals have 430 images. The unlabelled part comprises of 1,340 images. Figure 4.3 displays the distribution of WLT images from the labelled portion of the dataset

4.3 Methods

4.3.1 Model architecture

(a) Model architecture for HBW dataset:

The siamese network architecture used (Bromley et al., 1994; Chopra, Hadsell, and LeCun, 2005; Taigman et al., 2014) consisted of two parts, the base network for feature extraction and the matching network for computing the matching probability between extracted features. The base network uses a pair of CNNs which share the same parameters to summarise pairs of images. Practically, rather than keep two copies of the same CNN, only a single base CNN is used twice to extract feature vectors from both images. However, the execution time is doubled. The base model is a customer model originally inspired from deep residual networks referred to as ResNets (He et al., 2016a). ResNet architectures comprise many stacked residual blocks (He et al., 2016a; He et al., 2016b) each one expressed as follows: $\mathbf{y} = \mathcal{F}(\mathbf{x}, W_i) + h(\mathbf{x})$, where $\mathcal{F}(\mathbf{x}, W_i) = h(\mathbf{x}) - \mathbf{x}$ is the residual

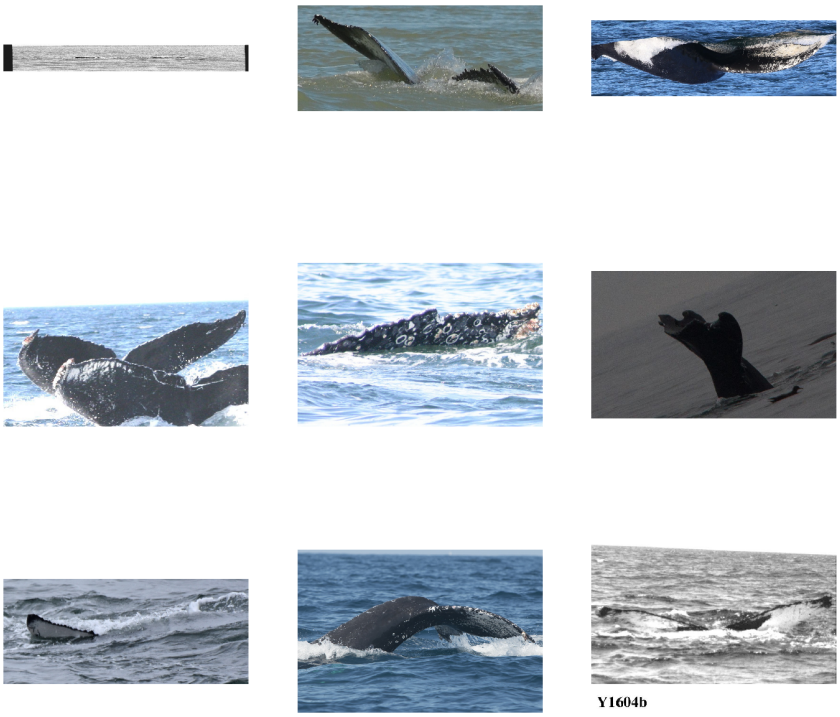


Figure 4.1: A sample of 10 HBW images selected from the list of ambiguous images.

mapping to be learned, and $h(\mathbf{x}) = \mathbf{x}$ is the identity mapping. The core strategy of ResNets is to learn an additive residual function $\mathcal{F}$ referenced to $h(\mathbf{x})$ via an identity mapping $h(\mathbf{x}) = \mathbf{x}$. The process is performed by means of an identity skip connection or short-cut. It is worth noting that the dimensions of $\mathcal{F}(\mathbf{x})$ and $\mathbf{x}$ have to match. Figure 4.4a is the residual block construction and Figure 4.4b is a bottleneck residual module with skip connection, where 1×1 convolutions are designed to decrease and restore the dimensions depending on the number of times this 1×1 convolution is performed. A small number allows for a decrease of the number of feature maps while a large number allows for an increase of the number of feature maps. The idea is to allow for the 3×3 convolution to remain with smaller input/output dimensions in order to reduce the computational cost but keeping the feature saliency.

The matching network combines image summarises into a similarity measures which can be used to predict the matching probability. The judgement concerning whether two images are taken from the same individual is made by setting a threshold probability beyond which a pair is a match otherwise a non-match. Figure 4.5 shows the diagram flow of the siamese architecture.

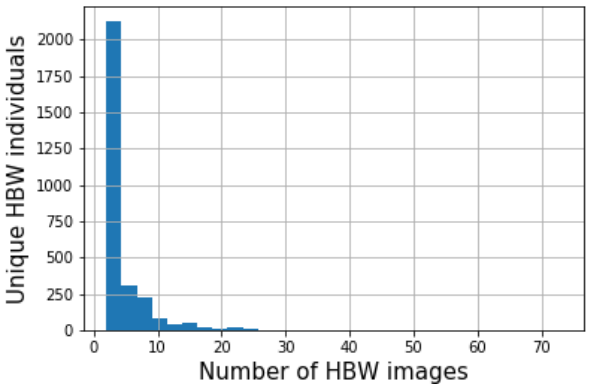


Figure 4.2: Distribution of HBW images

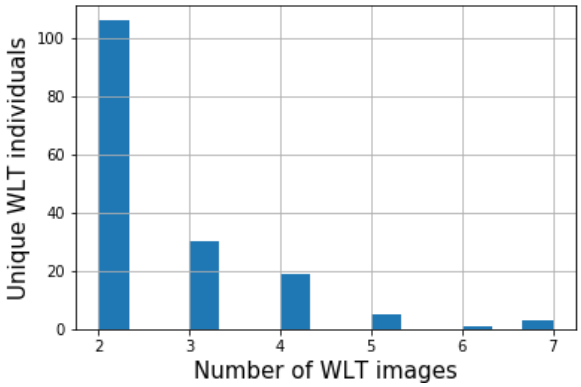
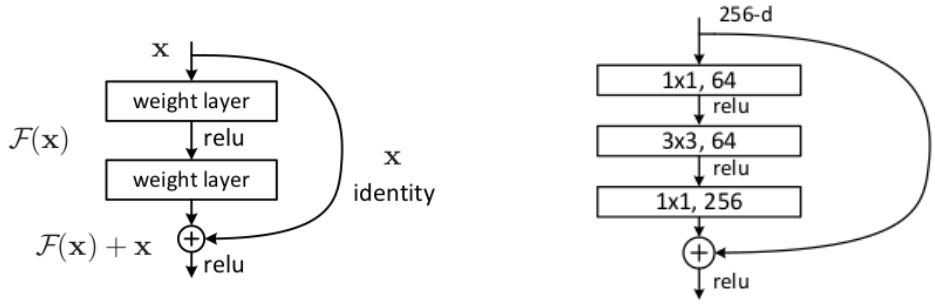


Figure 4.3: Distribution of WLT images

Base model: The base model is a residual neural network also called ResNet (He et al., 2016b) constructed taking into account the following ideas: Since the training dataset seems to be relatively small, the number of model parameters to be learned is kept relatively small. However, the model is made expressive enough. The base model consists of 6 convolutional blocks which process maps with lower and lower resolution (see Table 4.1 for more details), with intermediate pooling layers. The first block utilises a single convolutional layer with a kernel size of 9 and a stride of 2. It is followed by a 2×2 max pooling. Due do its higher resolution, it utilises large amounts of memory. As a result, subsequent layers are designed in a way to save memory. The second block is designed as a VGGish (Simonyan and Zisserman, 2014b) block with two 3×3 convolutions. These kind of convolutions are used to save memory since they are less memory intensive than the next ResNet blocks. The remaining are ResNet like blocks.

It is a common architecture design pattern that the number of filters often increases with the depth of the model (He et al., 2016a; Simonyan and Zisserman, 2014b). This scenario can lead to an increase of the model parameters and computations especially if larger kernel sizes are used. Though a pooling layer can be used to down sample the height and the width of the feature map by keeping salient features, it does not reduce the number of feature maps. To be able to control the depth of the network, ResNet blocks with 1×1 convolutions were used. The key idea about the construction of these blocks (He et al., 2016a) is the use of a sub-block with 1×1 convolution aiming at reducing the number of feature maps for computational efficiency purpose while retaining



(a) Building block

(b) Bottleneck residual module

Figure 4.4: Building block and a bottleneck residual module with skip connection (He et al., 2016a)

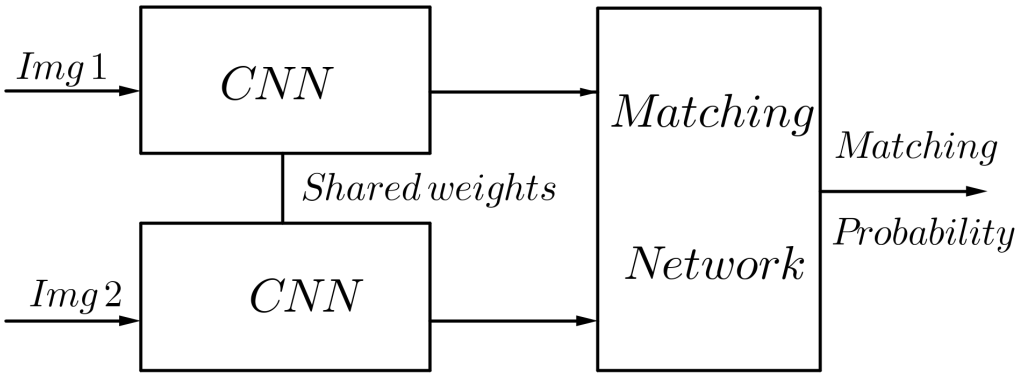


Figure 4.5: Siamese network architecture for computing the matching probability.

the saliency of the feature maps. It follows a 3×3 convolution to extend the receptive field to be able to capture the locally spatial information. Finally, a 1×1 convolution is used to restore the number of feature maps to the original. The result is then added to the original tensor by means of the skip connection. Each block utilised 4 sub-blocks and an additional 1×1 convolution computed after the pooling layer in the perspective of increasing the dimension of feature maps (Szegedy et al., 2015; He et al., 2016a). The **Base model** ends with a global max pooling layer to make extracted features more robust.

Matching netwok: The key idea behind this part of the model is to create a number of distance metrics and pass all of these to some dense layers to decide which, if any, are predictive of matching images. For each pair of feature vectors $(\boldsymbol{a_1}, \boldsymbol{a_2})$ extracted using the base model the following distance metrics are computed – the sum $(\boldsymbol{a_1} + \boldsymbol{a_2})$, the absolute difference $|\boldsymbol{a_1} - \boldsymbol{a_2}|$, the difference squared $(\boldsymbol{a_1} - \boldsymbol{a_2})^2$, and the product $(\boldsymbol{a_1 a_2})$. The four quantities are concatenated in a single vector and fed into a dense layer to decide their predictive importance. A sigmoid activation function is used at the output layer to convert the output into the matching probability.

(b) Model architecture for WLT dataset

The models fitted on WLT dataset used the same architecture as defined above. Only

Layer name	Feature map size	53 - weighted layers
Conv1	384×384	$\begin{bmatrix} 9\times 9, & 64 & \text{stride } 2 \\ 2\times 2 & \text{max pool}, & \text{stride } 2 \end{bmatrix}$
Conv2	96× 96	$\begin{bmatrix} 3\times 3, & 64 \\ 3\times 3, & 63 \end{bmatrix}$
Conv3	48×48	$\begin{bmatrix} 1\times 1, & 64 \\ 3\times 3, & 64 \\ 1\times 1, & 64 \end{bmatrix} \times 4$
Conv4	24×24	$\begin{bmatrix} 1\times 1, & 64 \\ 3\times 3, & 64 \\ 1\times 1, & 64 \end{bmatrix} \times 4$
Conv5	12×12	$\begin{bmatrix} 1\times 1, & 96 \\ 3\times 3, & 96 \\ 1\times 1, & 96 \end{bmatrix} \times 4$
Conv6	6×6	$\begin{bmatrix} 1\times 1, & 128 \\ 3\times 3, & 128 \\ 1\times 1, & 128 \end{bmatrix} \times 4$
Matching network	1×512	similarity measures, 4-d fc, sigmoid

Table 4.1: Building blocks of the siamese network. The Conv blocks are used to extract features and the Matching network to match them and score their matching probability.

one model which fine-tuned a ResNet pre-trained on ImageNet (Deng et al., 2009) made use of the base of ResNet50 as defined by He et al. (2016a) for feature extraction. The matching network was kept as described above.

4.3.2 Sampling matched and unmatched pairs

Training a DNN from random weights for a classification task requires a few thousands of representative samples per class (Russakovsky et al., 2015; Schneider et al., 2019). Given the datasets at hand, it seems more challenging to train a model which can be able to learn in most of time from 2 images per individual to classify the HBWs in more than two and a half thousand categories and WLTs in more than a hundred categories. More than that, categorising new images whose categories are not part of the training categories would be a challenge. Thus, the composition of both datasets guided the choice of the siamese network architecture (Chopra, Hadsell, and LeCun, 2005). Rather than focusing on recognizing the categories of images, the algorithm is exposed to matched pairs and unmatched pairs. The model learns to discriminate images from different individuals and to relate ones from the same individual based on the individual-specific pattern or markings.

To avoid images from the same individual to appear in both training and validation/test dataset, unique individuals were divided into two groups, one for the training set and another for validation/test set. Siamese neural networks accept **pairs** of images as inputs. A dataset consisting of all pairs of images from the training dataset was constructed. This

approach leads to two types of imbalance: (a) Among the $p(p-1)/2$ created pairs (where p is the number of images in the training dataset), there is a large number of unmatched pairs compared to matched pairs. (b) Individuals with many images appear more often in created pairs than others. It is due to the imbalance among individuals since they have different number of images. For example, if an individual comprises of n images, it appears $n(n-1)/2$ times in the matched pairs dataset. If an image appears more often in the training dataset, the model risks being biased to recognising them instead of the pattern or individual-specific markings. The other issue is the computational cost required to process the large number of pairs resulting from the constructed data. Hence, to overcome these issues, a sampling algorithm was implemented. It sampled an equal number of matched and unmatched training pairs. Besides that, for each training epoch, each training image only appears 4 times, twice in matched pairs and the other two in unmatched pairs. The size of the sampled pairs is twice the number of training images. The implementation of the sampling algorithm is described below.

Matched pairs:

Given a list L of images from the same individual, the images inside the list L are shuffled to create another list R in a way that the same image does not appear in the same position in the list L and R. This is done to avoid having a pair of the same image when doing an element-wise pairing between images from the list L and R. This procedure yields a number of random matched pairs where each image only appears twice, once in the list L and another in the list R. Putting together all pairs from different individuals yields p matched pairs equal to the number of images in the training dataset.

Unmatched pairs:

Unmatched pairs are created in the same way as matched pairs. All training images are put in a single list L', another list R' of the same images is created by shuffling the images inside the list R'. However, for this case, an element-wise pairing is only allowed for images from different individuals. The pairing is performed through the algorithm described below.

In both datasets, some images from different individuals look alike though they are not the same. The model can learn to correctly identify look-alike pairs as unmatched pairs if trained on them as it acquires the ability of discriminating between matched and unmatched pairs. To be able to choose pairs where the images are similar to one another, but from different individuals, the current model state was used to get the matching probability between all pairs of images. The result is a symmetric matrix of similarities. A linear assignment algorithm was applied to randomly select unmatched pairs, with probability proportional to their similarity + a constant (η). The constant is used to control the selection of unmatched pairs. By increasing the value of the constant one can flatten out the probability so that all pairs are equally likely to be selected. In order to select different unmatched pairs for consecutive epochs, entries selected in the previous epochs are set to $-\infty$ in the similarity matrix to tell the linear assignment algorithm to consider other choices for the next selection. The linear assignment algorithm returns p unmatched pairs equal to the total number of training images, wherein each image appears only once in the list L' and another in the list R'.

(a) HBWs training, validation, and test pairs:

The HBW dataset was divided into training and validation/test datasets. The training dataset consisted of 2,500 HBW unique individuals comprising 11,533 images. These

images generated 66,499,278 pairs of images with 58,195 matched and 66,441,083 unmatched pairs. The algorithm described above was used to sample 23,066 pairs of images after each 5 epochs. Half of them were matched pairs and another half unmatched pairs, and each image only appears 4 times in these pairs.

The validation and test dataset were drawn from 415 unique individuals with 1,929 images. These images create 1,859,556 pairs with 10,008 matched pairs and 1,849,548 unmatched pairs. 5,600 pairs were selected and 2,000 pairs with 964 matched and 1,036 unmatched pairs were used to validate the models. The remaining 3,667 pairs with 1,722 matched and 1,945 unmatched pairs were used for the test dataset. To get the overall model performance on the training dataset, the model was evaluated on 26,000 pairs of images, a half of them were matched and another half unmatched pairs randomly selected from training pairs. The first row of Figure 4.6 displays a matched pair and the second an unmatched pair. They are drawn from pairs sampled using the sampling algorithm described above.

(b) WLTs training and validation pairs:

The WLT dataset was split into training and validation datasets. The training dataset consisted of 150 unique WLT individuals with 388 images. These yield 75,078 pairs of images with 485 matched pairs and 74,593 unmatched pairs. The sampling algorithm was used to choose 778 pairs, a half matched and another unmatched pairs after each five epochs. The validation dataset had 14 unique individuals comprising 42 images. These images produces 861 pairs with 53 matched and 808 unmatched. From these, 90 pairs were used to validate the model. The first row of Figure 4.7 displays a matched pair and the second an unmatched pair of WLTs. They are selected using the sampling algorithm.

4.3.3 Image preprocessing

(a) HBW image preprocessing:

The majority of HBW images have a height and with greater than 250 pixels. The mean height and width are 505.45 and 1,000.12 respectively. Figure 4.8a and Figure 4.8b display HBW image height and width distributions. On one hand, resizing the image to the lower resolution leads to the loss of finer features and degrades the model performance (Koziarski Michałand Cyganek, 2018). On the other hand, keeping a large size of images leads to large amount of parameters which make the model hard to train. In this application, images are resized to a slightly higher resolution in the perspective of capturing some finer features like HBW tail fins and special marks. Some images were grayscale and others RGB. The solution was to convert all images into grayscale. An affine transformation (Weisstein, 2004) was then used to compress the image to the mean ratio and map its rectangular area to a square area of 384×384. The same resolution has been used in different applications (Simonyan and Zisserman, 2014a). At training time, online data augmentation is performed via an affine transformation which randomly composes rotation, shift, shear, and zoom to avoid original images appearing the same throughout the training. More importantly, data augmentation is not performed for validation and test cases, because these cases have to reflect the real data the model is likely to encounter when deployed in the real-world application. Images were normalised to zero mean and unit variance to reduce the memory and computation cost during training.

(b) WLT image processing: Many images in WLT dataset have a height and width greater than 500 pixels. The mean height and width are respectively 1,042.84 and 1,094.55 pixels. Figure 4.9a and Figure 4.9b display WLT image height and width distributions.

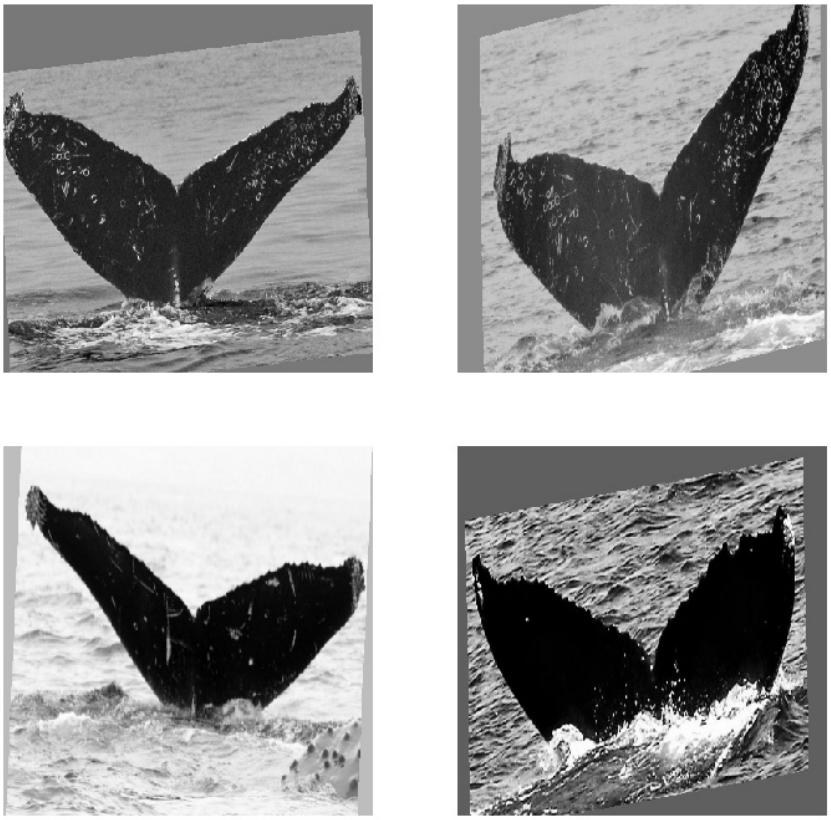


Figure 4.6: Two HBW training pairs, the first row is a matched and the second an unmatched pair.

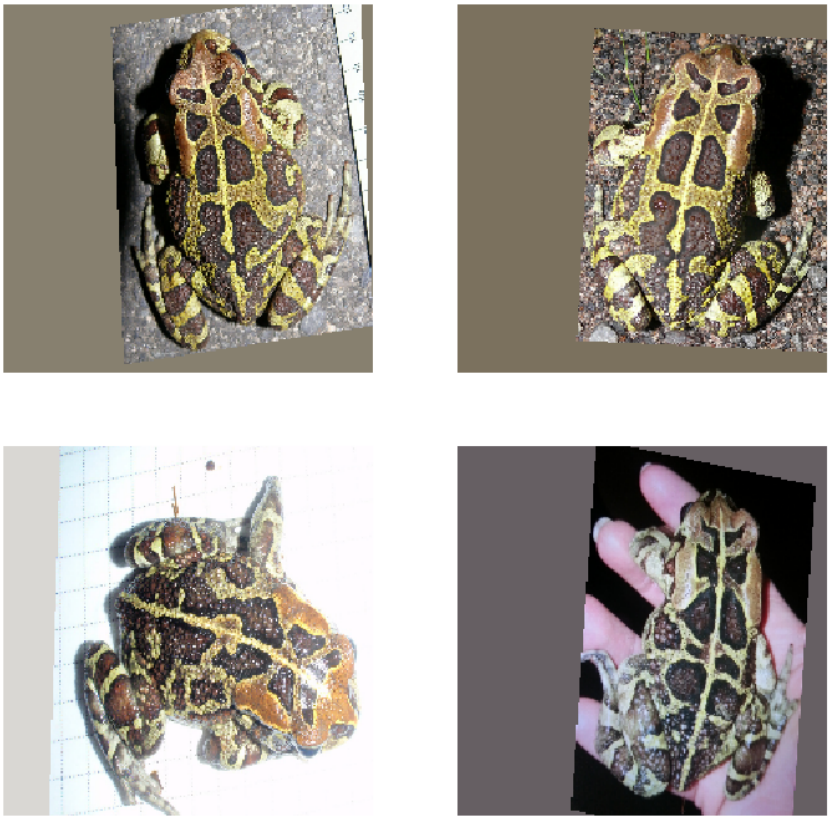
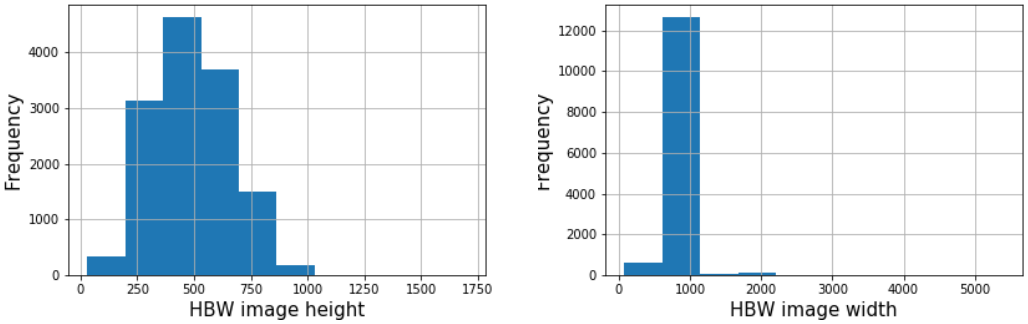


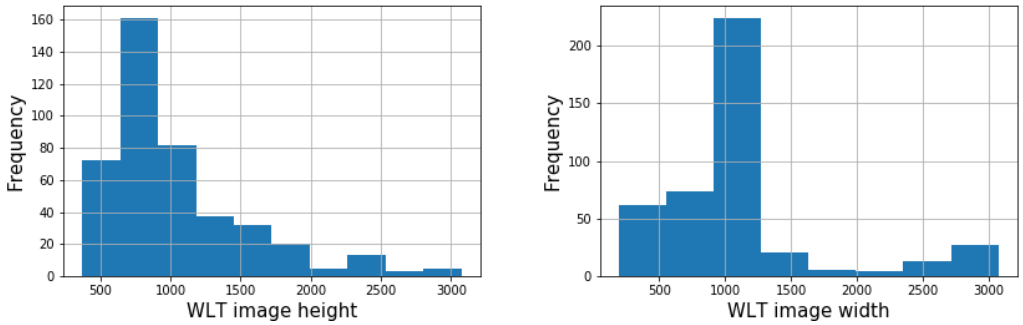
Figure 4.7: Two WLT training pairs, the first row is a matched and the second an unmatched pair.



(a) (b)

Figure 4.8: HBW image height and width distributions

This dataset applies the same steps used to pre-process HBW images. The only difference is that it utilises different image sizes to meet the requirements of the architectures used to train the models to avoid the discrepancy between the weights of the pre-trained model and the freshly initialised model. To be able to fine-tune the best HBW model on WLTs, the model used the same image size 384×384. Images were resized to 224×224×3 to be able to fine-tune a ResNet model trained using ImageNet (Deng et al., 2009) on WLT dataset. The same size has been use for many applications and architectures (Szegedy et al., 2015; He et al., 2016a; Simonyan and Zisserman, 2014a).



(a) (b)

Figure 4.9: WLT image height and with distributions

4.3.4 Model assessment

The following metrics were used to assess the performance of the models on both HBW and WLT datasets: accuracy, sensitivity, specificity, AUC, and F1 score.

4.3.5 Models and hyper-parameter selection

This section describes the models fitted on both datasets, the hyper-parameters involved in the training process, and the semi-supervised experiments carried over the WLT dataset.

(a) HBW model and hyper-parameters:

The model was run over 100 epochs and the Adam optimising algorithm with a mini-batch size of 32 was used to update the model parameters. The learning rate was 64×10^{-5} throughout the training. The parameter η controlling the selection of unmatched pairs was set to 1000 for the first 10 epochs. Afterwards, it was set to 100 and reduced by a factor of $100^{-0.1}$ each five epochs. The state of the best model was saved each time the validation loss is reduced. The last saved model is referred to as the final model and utilised to make predictions on the test dataset. It is worth mentioning that an exhaustive search of all possible hyper-parameters was not done due to the computational resources limitations. Only some values were experimented with to get a combination of which leads to reliable results. Each epoch took 9 min on average using the g3.4xlarge Amazon Web Service (AWS) spot instances (Section 4.3.6).

(b) WLT models and hyper-parameters:

Four models were fitted on WLT dataset in order to select the best model used to carry out the semi-supervised experiments.

1. The first model named **ScratchRGB** in Table 3.6 is trained from random weights on RGB images of size 224×224×3 by randomly initialising weights and optimising them through an iterative process during training.
2. The second model referred to as **ScratchGrayscale** in Table 3.6 is also trained from randomly initialised weights. It is trained on grayscaled images of size 224×224. This model is fitted in the perspective of checking if the model performs better on grayscale images than RGB images.
3. The third model named **ResnetFinetuned** in Table 3.6 fine-tuned ResNet50 (He et al., 2016a) model trained on ImageNet (Deng et al., 2009). The convolutional part of the ResNet50 is used for feature vector extraction. The model is trained on RGB images of the same size 224×224×3 as ResNet50. The motivation behind fitting this model is to check if the model can utilise the knowledge obtained from categorising different images from ImageNet dataset to identify WLTs. Convolutional neural networks learn more generic features in the earlier layers and deal with complex objects in later layers. To leverage low and mid-level features learned from ImageNet and generalise on the WLT dataset, the weights in the first 78 layers are initialised to the values obtained using the ImageNet dataset. These weights are not estimated, they are held fixed. Weights in the remaining layers are estimated.
4. The last model referred to as **WhalesFinetuned** in Table 3.6 fine-tuned the model trained on HBWs. It is trained on grayscale images of size 384×384. The objective of this model is to test if the model is able to utilise the knowledge gained from identifying HBWs to identify WLTs. To benefit from low and mid-level features learned from the HBW dataset, the weights in the first 4 convolutional blocks (Table 4.5) are not estimated, they are held fixed and used to estimate the remaining weights.

The four models were run over 100 epochs. The Adam optimiser was used with a mini-batch size 32 to update the model parameters during training. A learning rate of 64×10^{-5} was used for the two first models and 32×10^{-5} for the last two models. The parameter η controlling unmatched pairs selection was set to 1,000 for the first 10 epochs. To allow the selection of similar looking unmatched pairs, for the rest of the training η was set to 100 and was being reduced by a factor of $100^{-0.1}$ for each five epochs. During the training, the best model was saved each time the validation loss function drops. The last saved model is referred to as the best model. It is worth mentioning that an exhaustive search

of all possible parameters was not performed due to computational resources limitations. The process experimented with some values to get a combination of hyper-parameters which leads to reliable results. Each epoch took on average 2 min using the South African Centre for Higher Performance Computing CHPC (see Section 4.3.6).

(c) WLT semi-supervised learning experiments:

Since some individuals in the training and validation datasets may still have some images in the unlabelled dataset, a single image representing training and validation individuals was added onto unlabelled dataset to be able to capture their remaining images if any. The WLT best model was then run over the unlabelled dataset once and obtained predicted probabilities of a match for every pair of images. 100 image pairs with the highest probability of a match were manually checked. Of these, some were true matches (correctly identified) and some were not (incorrectly identified). All individuals involved in the true matches were extracted and added to the training dataset. Due to the way the training pairs are constructed (see Section 4.3.2), these new images will appear in both matched pairs and unmatched pairs. After adding the newly identified individuals to the training images, the model retrained and the checking process is then repeated.

4.3.6 Package and software

The following two paragraphs summarise the computational resources, software, libraries, and modules involved in the implementation of the models.

(a) Computational resources, libraries and software used for HBW models:

Processing images involves arrays that utilise a large amount of memory. One needs large memory storage and efficient computational resources. The g3.4xlarge Amazon Web Services (AWS) spot instances from the Amazon Elastic Computing Cloud known as EC2 was used to process the HBW models. It is one of the suitable instances for deep learning models. It has 1 GPU with a GPU memory support of 8 GB, 16 vCPU, with the main memory of 122 GB and a large internet bandwidth of 3.5 Gbps. The Python data analysis library Pandas was used to read and manipulate .csv files. Images were loaded and visualised in the working environment using Python Image Library (PIL). An affine transformation function from SciPy the Python scientific computing library was used through image preprocessing phase. The linear assignment problem solver referred to as LAP module was used through the implementation of the sampling algorithm used to sample matched and unmatched pairs. Some metrics from the scikit-learn library were used to assess the model performance. Models were implemented in Keras deep learning framework using Python programming language and Tensorflow back-end.

(b) Computational resources, libraries and software used for WLT models:

The South African Centre of High Performance Computing CHPC platform also known as Lengau cluster was used for computational resources. A memory storage of 125 GB was available. Each model was submitted as a job in the queue and the scheduler assigned it to a compute node of 24 cores, 24 CPUs, and 128 GB of memory. The rest was implemented as indicated above in (a).

4.4 Results

4.4.1 Model results and discussion on HBW dataset

Table 4.2 summarises the model performance on the HBW individual identification task over the training, validation, and test pairs as described under Section 4.3.2. In 95% of the cases, the model was able to correctly identify from the validation and test pairs that a pair of images was drawn from the same individual or not. Some images looked similar though they were from different individuals. This idea led the choice of introducing in the model an algorithm which sampled similar looking pairs from different individuals to force the algorithm to consider them as unmatched pairs. The results in Table 4.2 show that the model achieved a slightly higher specificity on the validation and test datasets compared to sensitivity. It correctly identified 96% and 97% of unmatched pairs in the validation and test pairs respectively. In contrast, it correctly identified 95% of matched pairs in the validation dataset and 93% in the test dataset. During earlier iterations of the training process, the performance on the validation dataset were slightly higher compared to the performance on the training dataset. This was due to the fact that the model was being trained on hard unmatched pairs (looking similar unmatched pairs). In the later iterations, once the model has trained enough on hard unmatched pairs, its training performance increased. On average, the five metrics show that the model achieved a very good and stable performance on the HBW individual identification task. The confusion matrices Table 4.3a and Table 4.3b show respectively the detailed predictions on the training and test pairs. Figure 4.10 and Figure 4.11 display respectively examples of correctly identified and misidentified HBW test pairs.

Metrics	Training	Validation	Test
Accuracy	0.97	0.95	0.95
Sensitivity	0.98	0.95	0.93
Specificity	0.97	0.96	0.97
AUC	0.97	0.95	0.95
F1 score	0.97	0.95	0.95

Table 4.2: Model performance on HBW individual identification task. The model is assessed by the accuracy, sensitivity, specificity, AUC and F1 score metrics.

Obs \ Pred	Unmatched	Matched
Unmatched	12,630	370
Matched	319	12,681

(a) Training

Obs \ Pred	Unmatched	Matched
Unmatched	1,891	54
Matched	120	1,602

(b) Test

Table 4.3: Training and test confusion matrices. Obs and Pred stand respectively for observed and predicted values.

4.4.2 Model results and discussion on WLT dataset

Table 4.4 summarises the performance of four models on the WLT individual identification task over the validation pairs. In 87% of the cases, the three first models in Table 4.4 correctly identified whether a pair of images is from the same WLT individual

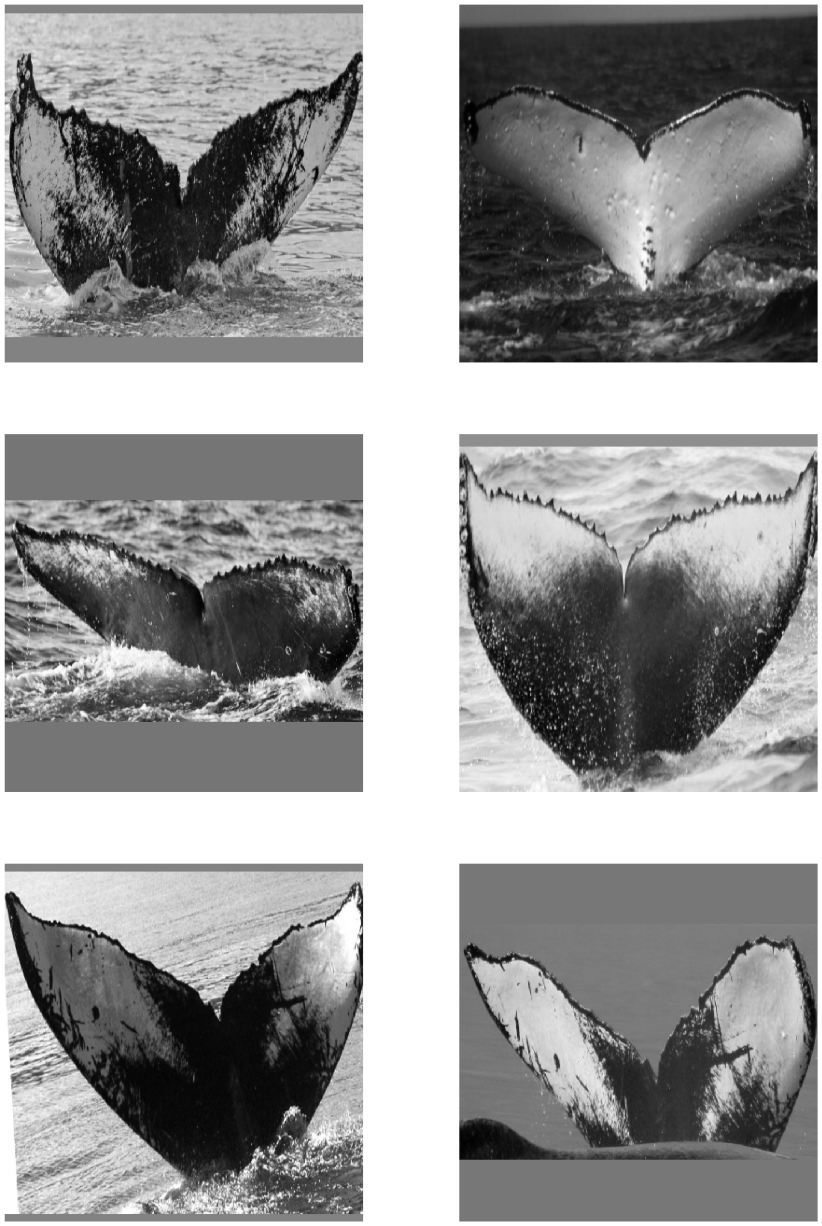


Figure 4.10: Examples of correctly identified pairs. The first row is an unmatched pair and the remaining are matched pairs.

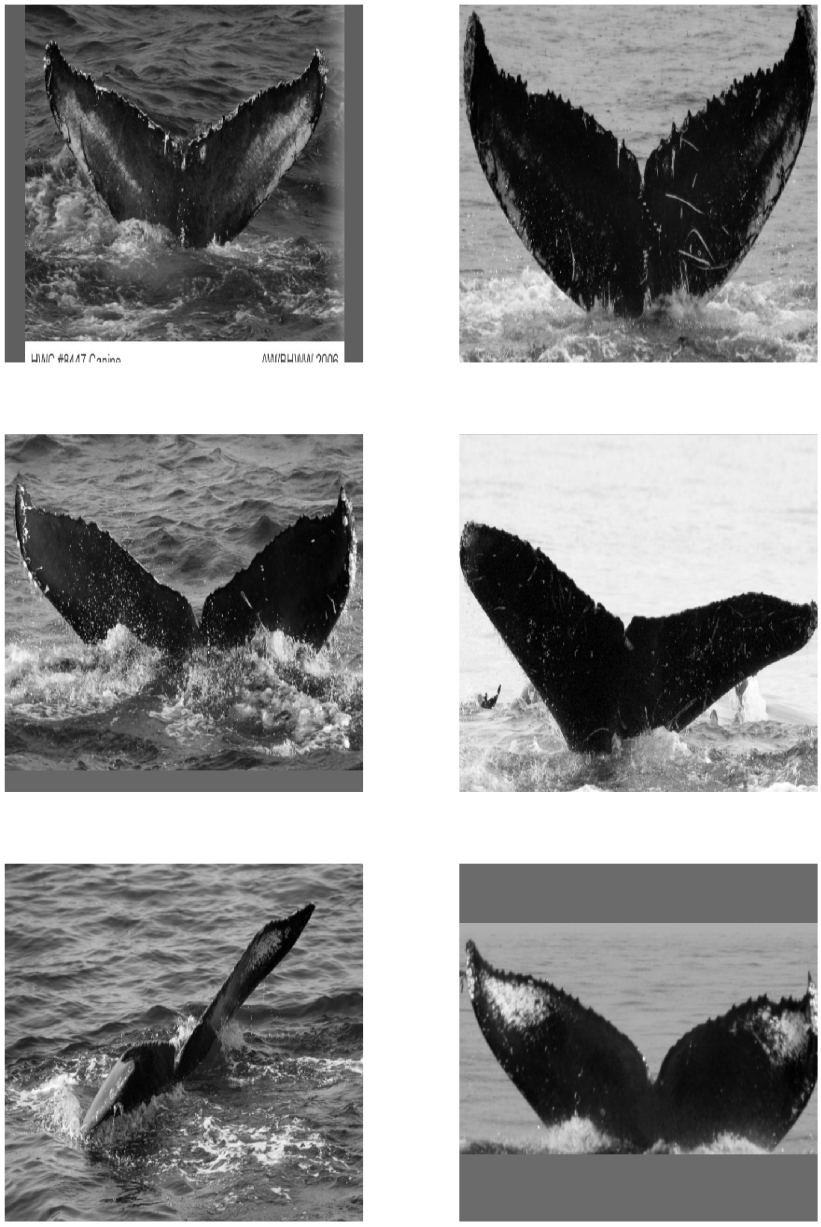


Figure 4.11: Examples of misidentified pairs. The first row was a matched pair but misidentified as an unmatched pair by the model. The remaining were unmatched pairs but misidentified as matched pairs.

or not. Nevertheless, these models have different ability in identifying matched pairs (sensitivity) and unmatched pairs (specificity). The first two models were likely to correctly identify unmatched pairs compare to matched pairs, while the third model showed the identical ability on both matched and unmatched pairs. The model **ScratchRGB** and **ScratchGrayscale** trained from random weights on RGB and grayscale images achieved almost identical results. The model **WhalesFinetuned** fine-tuned from the best model on HBW identification task achieved lower performance among the four fitted models. It shows that the model failed to leverage the knowledge gained from HBW individual identification task. It might be due to the difference between key features, spots for WLTs and the shape of HBW tail and special marks for HBW. The model **ResnetFinetuned** fine-tuned from ResNet50 pre-trained on ImageNet achieved stable results for all five metrics. This might be due to the fact that the ResNet50 was pre-trained on ImageNet (Deng et al., 2009) containing 1000 different categories including various types of toads. During training, the fine-tuned models (**ResnetFinetuned** and **WhalesFinetuned**) converged after a few iterations compared to models trained from random weights. This can be explained by the fact that these models were only training a few parameters while maintaining a large amount of parameters fixed. The **ResnetFinetuned** model was selected to carry out the semi-supervised experiments. The confusion matrix Table 4.5 gives more details on **ResnetFinetuned** model predictions over the validation dataset. Figure 4.12 and Figure 4.13 display respectively the examples of accurately identified and misidentified pairs.

Model	Accuracy	AUC	Specificity	Sensitivity	F1 score
ScratchRGB	0.87	0.87	0.98	0.75	0.88
ScratchGrayscale	0.87	0.87	0.96	0.78	0.88
ResnetFinetuned	0.87	0.87	0.87	0.87	0.87
WhalesFinetuned	0.81	0.67	0.96	0.38	0.73

Table 4.4: Performance of four models on WLT individual identification task over the validation dataset. Models are assessed using the accuracy, AUC, sensitivity, specificity, and F1 score metrics.

4.4.3 Result from semi-supervised experiments

After 5 rounds of semi-supervised experiments, the model managed to identify 47 new matched pairs in the following order: 8, 9, 13, 10, and 7. Some matched pairs from the same individual were identified through different rounds. After extracting and grouping them according to their individuals, the model identified up to 26 new individuals comprising 63 images. After adding these newly identified individuals to the training set, the model slightly improved its results. It achieved a validation accuracy of 0.89 over 120 pairs. It is noteworthy mentioning that the model was being trained over images from

Obs \ Pred	Unmatched	Matched
Unmatched	39	6
Matched	6	39

Table 4.5: Validation confusion matrix computed from the predictions of **ResnetFinetuned** model. Obs and Pred stands for observed and predicted values.

Figure 4.12: Examples of correctly identified WLT pairs. The first row is an unmatched pair and the last is a matched pair.

Figure 4.13: Examples of misidentified pairs. The first row was un unmatched pair but misidentified as a matched pair and the last was a matched pair but misidentified un unmatched pair.

Obs \ Pred	Unmatched	Matched
Unmatched	48	8
Matched	4	52

Table 4.6: Validation matrix computed from 120 WLT validation pairs. Obs and Pred stand for observed and predicted values.

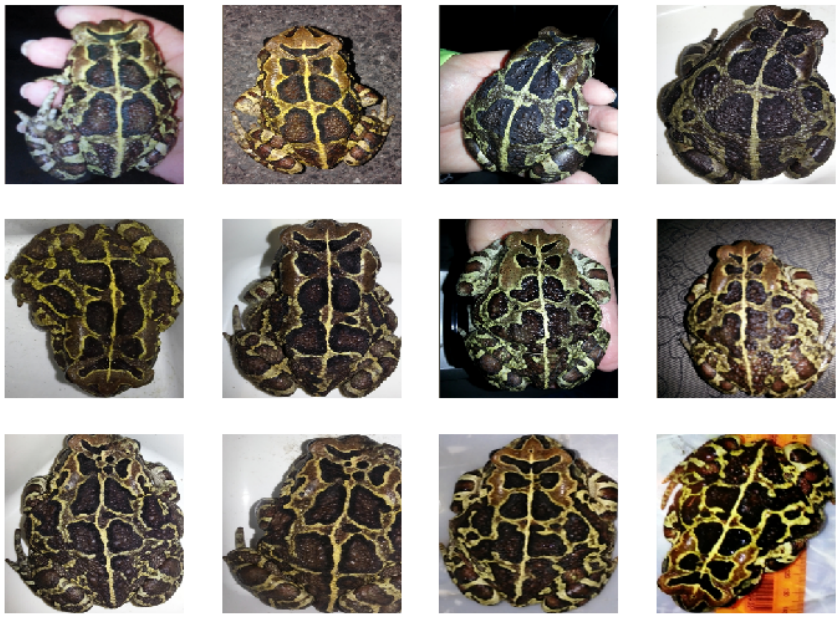


Figure 4.14: A sample of identified matched pairs through semi-supervised experiments. For each row, the first two and last two images form matched pairs.

less than 180 unique individuals. In other words, it was being trained over almost 180 patterns. However, when applied over almost 1,400 images, it is likely to encounter less or more than a thousand of different unique patterns with various shapes. As a result, a model trained on almost 180 patterns might not generalise well on a thousand patterns. The confusion matrix Table 4.6 gives detailed predictions from 120 validation pairs. Figure 4.14 displays a sample of matched pairs identified through the semi-supervised learning process. For each row, the first two and the last two images form matched pairs. Figure 4.15 displays a sample of some look-alike unmatched pairs that the model assigned a high matching probability. For each row, the first two and the last two images form look-alike unmatched pa

4.5 Discussion

The models were able to achieved good results for the HBW and WLT individual identification task. However, the performance of the model was higher on the HBW case

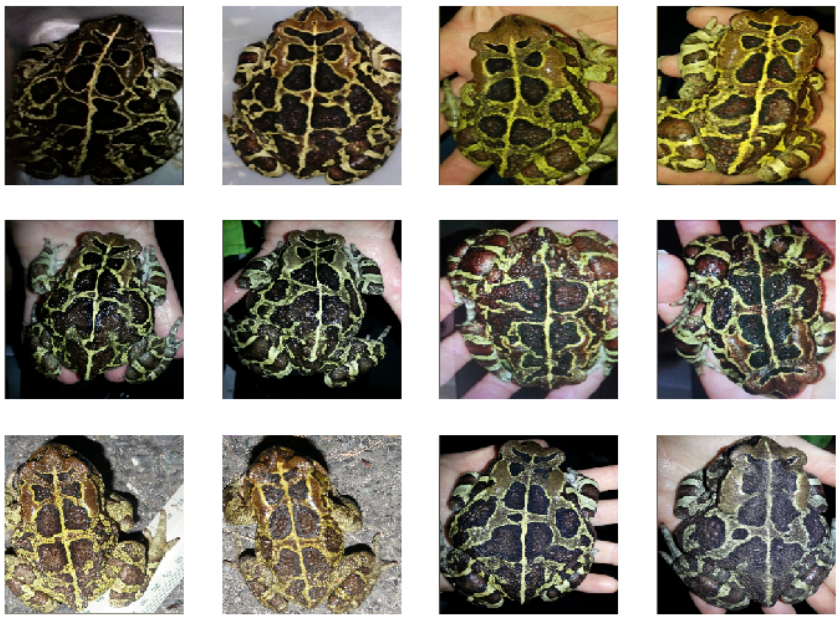


Figure 4.15: A sample of some look-alike unmatched pairs that the model assigned a high matching probability. For each row, the first two and the last two images form unmatched pairs.

compared to the WLT case. The difference in performance is mainly due to the fact that the HBW dataset had many images to train the model whilst the WLT dataset had a few images. Improving the model performance further might require getting more data. The semi-supervised approach was partially successful in this application. The model was able to identify new individuals and matches that would otherwise not have been detected, but these were relatively few in numbers. Without an exhaustive check of the data, it is not clear whether this is due to a failure of the semi-supervised approach, or because there are not many matches in the data.

Model usage:

The model can be used in two slightly different scenarios. Either it can be used to identify new matches in a set of new individuals or to classify a new image into one of the existing individual classes. In the former scenario, the new images are paired to create pairs. The model is then used to predict the matching probability on these pairs which is closer to 1 if two images are from the same individual and 0 otherwise. In the latter scenario, a new image is paired with images (one per individual) representing existing individual classes. The model computes the matching probability for all formed pairs. The winning individual class is the one with the highest probability, closer to 1. If all probabilities are closer to 0, this implies that there is no match in the dataset. As a result, the new image is classified as a new individual.

It is worth noting that the model output is a probability. The decision concerning whether a pair is a match or not is made by setting a probability threshold beyond which a pair is a match otherwise a non-match. A probability threshold of 0.5 was used during training and validation/test phases.

4.6 WildID software results

The Wild-ID software (T. Bolger et al., 2012) utilises the Scale Invariant Feature Transform (SIFT) operator (Lowe, 2004) for the extraction of distinctive image features. Given a new image, the algorithm inspects if it has a match by computing the SIFT features of all images in the database, comparing them with the SIFT features of the new image in a pairwise fashion and computing the corresponding matching scores. The algorithm ranks 20 potential image matches from the database associated with the top 20 higher matching scores. The user examines the ranked images and confirms either among them there is a match or not. It is worth mentioning that: (a) The Wild-ID software processes a single image at a time (images ranked alphabetically). After the comparison of the new image with the existing images, the database is updated by adding the newly processed image. (b) Though the algorithm ranks up to 20 images, it only allows the user for confirming a single match. Once the user has confirmed a match, the software records it in a particular file in the database, ignores the remaining and moves onto the next image to be processed. If there is no match, the user rejects all potential candidates and Wild-ID processes the next image.

4.6.1 Wild-ID data preparation

The Wild-ID software is tested over 300 images drawn from 250 different individuals. 200 among the individuals have a single image and the remaining 50 have two images which allow for having at least a match in the dataset. Wild-ID processes images according to

their alphabetical order. As a result, the individuals without a match in the dataset are assigned names which allow them for being processed before individuals with a match in the dataset to facilitate the manual check. The images are loaded and processed in the following order:

a) The 200 different individuals which do not have any matches in the dataset are processed first.

b) They are followed by 50 images from different individuals which have a match in the dataset. These are still different individuals, none of their matches is included yet.

c) Finally, 50 matches one for each individual loaded in b) are loaded and processed.

Though the algorithm processes and ranks potential matches for a) and b), there is no use in checking them since these images are known to have no matches yet. Each time, all ranked candidates are rejected to allow for the user to move onto the next image. All images in c) have a match in the database from b) and are expected to appear high up in the ranked potential matches.

4.6.2 Wild-ID results and discussion

a) Wild-ID result on the HBW dataset:

The Wild-ID achieved slightly lower performance on HBW dataset. It was only able to rank the match among the top 20 images in $28/50 = 56\%$ of the cases. Table 4.7 gives more details on where the match appeared among the top 20. The Absence option means that the match did not appear among the ranked images.

Rank	Number of matches
[1-5]	22
[6-10]	2
[10-15]	3
[15-20]	1
Absence	22

Table 4.7: Rank of 50 HBW matches by the Wild-ID software. The Rank column displays the interval wherein the match appeared in the top 20 ranked images. In the 28/50 of the cases, the match appeared in the first 20 ranked images.

b) Wild-ID result on the WLT dataset:

The Wild-ID software achieved good result on the WLT dataset. In $43/50 = 86\%$ of the cases the software managed to rank a match among the top 20 images. Table 4.8 gives more details on where the match appeared among the top 20. The Absence case means that the match did not appear among the top 20 ranked images.

4.6.3 Discussion

Wild-ID achieved good result on the WLT dataset compared to the HBW dataset. For both datasets, in many cases the match appeared in the first top 5 candidates. There is no a direct way to compare the model and Wild-ID results since the former uses the

Rank	Number of matches
[1-5]	38
[6-10]	3
[10-15]	2
[15-20]	–
Absence	7

Table 4.8: Rank of 50 WLT matches by the Wild-ID software. The Rank column displays the interval wherein the match appeared in the top 20 ranked images. In the 43/50 of the cases, the match appeared in the first 20 ranked images.

top 1 accuracy and the later the top 20 accuracy. However, with much less effort made, it can be concluded that the model performance is comparable on the WLT dataset and higher on the HBW dataset compared to the Wild-ID results.

Chapter 5

Conclusions

Wildlife conservation measures are taken based on information about existing demographics, for example population sizes and population growth rates. Monitoring species requires one to observe them over time by means of methods which allow their recognition. Capture-mark-recapture is the common technique used in ecology for individual identification (Whitehead, Christal, and Tyack, 2000; Auckland, Debinski, and Clark, 2004; Watkins et al., 1993). However, this technique can be harmful in some cases where the marking is invasive and the marking events sometimes disrupt the wildlife (Weinstein, 2018).

This thesis developed a machine learning algorithm which exploits individual-specific markings as well as latent features to automate the individual identification task. It assessed the effectiveness of the semi-supervised approach on WLT unlabelled dataset and compared the siamese network and Wild-ID (one of the computer-aided photo-matching algorithms) results on individual identification task. After reviewing the theory behind the semi-supervised learning and neural network algorithms, the thesis built an automated image matching model which consisted of two main tasks – identifying the region of the image containing the animal (bounding box detection), and classifying two images within these identified bounding boxes as originating from the same individual (identification).

The bounding box detection task was motivated by the fact that in some images the animal only occupied a little space, for example a species inside an object. This caused the pattern on images – spots on the back of the WLTs, and shape of HBW tail fins and special marks to not be clearly visible while they are the key feature for the individual identification task. The task of manually extracting animals from thousands of photographs is tedious and time-consuming. As a result, this thesis implemented a semi-automated algorithm which used image thresholding and edge detection to generate bounding box data for a large number of training images. This is faster than doing it manually. It is noteworthy mentioning that there is no threshold which can allow for accurately cropping all images at once. Hence the algorithm was semi-automated rather than fully automated. Thereafter, the generated bounding box data was used to train a CNN to automate the detection and extraction of the animals from images. The models were assessed using the mean squared error **MSE**, the coefficient of determination $\mathbf{R}^2$, and the intersection over union **IoU** and achieved reliable results on both HBW and WLT datasets. The resulting models were able to reliably crop the remaining images which were not accurately cropped by the semi-automated algorithm and thus saving both effort and time. The cropped images were finally used to train a siamese network for individual identification. In 95% of the cases, the model was able to reliably identify whether two HBW images are from the same individual or not on previously unseen pairs. For WLT dataset, the model was able

to correctly identify if two toad images are from the same individual or not in 87% of the cases of previously unseen pairs.

The semi-supervised learning process was performed over the unlabelled part of the WLT dataset. In this project, the semi-supervised approach was partially successful. For a few rounds performed, the model was able to identify 47 new matches from 26 new individuals. It was not clear that the model got few matches and new individuals because the semi-supervised approach failed or because there were not many matches in the dataset. After adding the newly identified individuals to the WLT labelled dataset, the model slightly improved its performance and correctly identified 89% of WLT pairs.

The results showed that the model trained on HBWs achieved higher performance. This is due to the fact that it was trained on a relatively bigger dataset ($>$ 10,000 images) which allowed the algorithm to explore more shapes of HBW tail fins and special markings for better generalisation. The WLT models were trained on about 500 images only. Improving the performance of these models almost surely requires more data.

The comparison between the model and Wild-ID (one of the existing computer-aided photo-matching algorithms) results showed that the model achieved very competitive results compared with Wild-ID. It has been shown that the model can be used in two slightly different scenarios. It can either be used to identify individuals in a new dataset or classify new images in one of the existing individual classes.

This project only focussed on photo matching based on individual-specific marks. Other information is often available, for example spatial locations and times where the photos were taken. For future work, these could be integrated into the analysis. Another topic for future work is to use the matches generated by the model for capture-recapture (CR). Currently CR methods assume a correct individual ID, while the model gives a probability of match, which gives important information about the uncertainty of a match. One possibility would be to treat all model predictions as correct, but this would almost certainly be wrong in some cases. Another much more difficult option is to adapt CR models so that instead of perfect individual capture histories, they take as input data the uncertain capture histories.

Bibliography

Agatonovic-Kustrin, S and R Beresford (2000). "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research". In: *Journal of pharmaceutical and biomedical analysis* 22.5, pp. 717–727.

Aggarwal, Charu C (2018). *Neural networks and deep learning*. Springer.

Anonymous (2018). "On the Convergence of Adam and Beyond". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=ryQu7f-RZ.

Ardovini, Alessandro, Luigi Cinque, and Enver Sangineto (2008). "Identifying elephant photos by multi-curve matching". In: *Pattern Recognition* 41.6, pp. 1867–1877.

Arzoumanian, Z, J Holmberg, and B Norman (2005). "An astronomical pattern-matching algorithm for computer-aided identification of whale sharks Rhincodon typus". In: *Journal of Applied Ecology* 42.6, pp. 999–1011.

Auckland, Julia N, Diane M Debinski, and William R Clark (2004). "Survival, movement, and resource use of the butterfly Parnassius clodius". In: *Ecological Entomology* 29.2, pp. 139–149.

Baldi, Pierre and Yves Chauvin (1993). "Neural networks for fingerprint recognition". In: *Neural Computation* 5.3, pp. 402–418.

Beijbom, Oscar et al. (2016). "Improving automated annotation of benthic survey images using wide-band fluorescence". In: *Scientific reports* 6, p. 23166.

Bengio, Yoshua (2012a). "Deep learning of representations for unsupervised and transfer learning". In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 17–36.

– (2012b). "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*. Springer, pp. 437–478.

Bishop, Christopher M and Others (1995). *Neural networks for pattern recognition*. Oxford university press.

Blanc, Katy, Diane Lingrand, and Frédéric Precioso (2014). "Fish species recognition from video using SVM classifier". In: *Proceedings of the 3rd ACM International Workshop on Multimedia Analysis for Ecological Data*. ACM, pp. 1–6.

Bolger, Douglas T et al. (2012). "A computer-assisted system for photographic mark–recapture analysis". In: *Methods in Ecology and Evolution* 3.5, pp. 813–822.

Booth, David J (2004). "Synergistic effects of conspecifics and food on growth and energy allocation of a damselfish". In: *Ecology* 85.10, pp. 2881–2887.

Bradley, Andrew P (1997). "The use of the area under the ROC curve in the evaluation of machine learning algorithms". In: *Pattern recognition* 30.7, pp. 1145–1159.

Bradshaw, Corey J A, Richard J Barker, and Lloyd S Davis (2000). "Modeling tag loss in New Zealand fur seal pups". In: *Journal of Agricultural, Biological, and Environmental Statistics* 5.4, pp. 475–485.

Branson, Steve et al. (2014). "Bird species categorization using pose normalized deep convolutional nets". In: *arXiv preprint arXiv:1406.2952*.

Bromley, Jane et al. (1994). "Signature verification using a" siamese" time delay neural network". In: *Advances in neural information processing systems*, pp. 737–744.

Caruana, Rich, Steve Lawrence, and C Lee Giles (2001). "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping". In: *Advances in neural information processing systems*, pp. 402–408.

Caughley, Graeme and Annecoaut Gunn (1996). *Conservation biology in theory and practice*. 333.9516 C3.

Chaki, Nabendu, Soharab Hossain Shaikh, and Khalid Saeed (2014). "A comprehensive survey on image binarization techniques". In: *Exploring Image Binarization Techniques*. Springer, pp. 5–15.

Chapelle, Olivier, Bernhard Scholkopf, and Alexander Zien (2009). "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]". In: *IEEE Transactions on Neural Networks* 20.3, p. 542.

Chen, Colin (2002). "Paper 265-27 robust regression and outlier detection with the robustreg procedure". In: *Proceedings of the Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference*.

Cho, Kyunghyun et al. (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*.

Chopra, Sumit, Raia Hadsell, and Yann LeCun (2005). "Learning a similarity metric discriminatively, with application to face verification". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE, pp. 539–546.

Choromanska, Anna et al. (2014). "The Loss Surface of Multilayer Networks". In: *CoRR* abs/1412.0. arXiv: 1412.0233. URL: http://arxiv.org/abs/1412.0233.

Christin, Sylvain, Eric Hervet, and Nicolas Lecomte (2018). "Applications for deep learning in ecology". In: *bioRxiv*, p. 334854.

Cordts, Marius et al. (2016). "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223.

Crall, Jonathan P et al. (2013). "Hotspotter—patterned species instance recognition". In: *2013 IEEE workshop on applications of computer vision (WACV)*. IEEE, pp. 230–237.

Cross, MATTHEW D et al. (2014). "Pattern-recognition software as a supplemental method of identifying individual eastern box turtles (Terrapene c. carolina)". In: *Herpetological Review* 45.4, pp. 584–586.

Cuthill, Innes (1991). "Field experiments in animal behaviour: methods and ethics". In: *Animal Behaviour* 42.6, pp. 1007–1014.

Dala-Corte, Renato B, Júlia B Moschetta, and Fernando G Becker (2016). "Photo-identification as a technique for recognition of individual fish: a test with the freshwater armored catfish Rineloricaria aequalicuspis Reis & Cardoso, 2001 (Siluriformes: Loricariidae)". In: *Neotropical Ichthyology* 14.1.

Dauphin, Yann et al. (2014). "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *CoRR* abs/1406.2. arXiv: 1406.2572. URL: http://arxiv.org/abs/1406.2572.

Deb, Debayan et al. (2019). "Face recognition: Primates in the wild". In: *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*. IEEE, pp. 1–10.

Dell, Anthony I et al. (2014). "Automated image-based tracking and its application in ecology". In: *Trends in ecology & evolution* 29.7, pp. 417–428.

Deng, Jia et al. (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Deng, Li, Geoffrey Hinton, and Brian Kingsbury (2013). "New types of deep neural network learning for speech recognition and related applications: An overview". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, pp. 8599–8603.

Dosovitskiy, Alexey et al. (2015). "Flownet: Learning optical flow with convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2758–2766.

Fushiki, Tadayoshi (2011). "Estimation of prediction error by using K-fold cross-validation". In: *Statistics and Computing* 21.2, pp. 137–146.

Gamble, Lloyd, Sai Ravela, and Kevin McGarigal (2008). "Multi-scale features for identifying individuals in large biological databases: an application of pattern recognition technology to the marbled salamander Ambystoma opacum". In: *Journal of Applied Ecology* 45.1, pp. 170–180.

Garcia, Noa and George Vogiatzis (2017). "Learning Non-Metric Visual Similarity for Image Retrieval". In: *arXiv preprint arXiv:1709.01353*.

Gatos, Basilis, Konstantinos Ntirogiannis, and Ioannis Pratikakis (2009). "ICDAR 2009 document image binarization contest (DIBCO 2009)". In: *2009 10th International Conference on document analysis and recognition*. IEEE, pp. 1375–1382.

Girshick, Ross (2015). "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448.

Girshick, Ross et al. (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep sparse rectifier neural networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323.

Gomez, Alexander et al. (2016). "Animal identification in low quality camera-trap images using very deep convolutional neural networks and confidence thresholds". In: *International Symposium on Visual Computing*. Springer, pp. 747–756.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Gori, M and A Tesi (1992). "On The Problem Of Local Minima In Backpropagation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14, pp. 76–86.

Han, Xufeng et al. (2015). "Matchnet: Unifying feature and metric learning for patch-based matching". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3279–3286.

He, Kaiming et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

– (2016a). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

– (2016b). "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer, pp. 630–645.

Hernández-Serna, Andrés and Luz Fernanda Jiménez-Segura (2014). "Automatic identification of species with neural networks". In: *PeerJ* 2, e563.

Hinton, Geoffrey et al. (2012). "Deep neural networks for acoustic modeling in speech recognition". In: *IEEE Signal processing magazine* 29.

Howland, Jonathan C, Nicholas B W Macfarlane, and Peter Tyack (2012). "Precise geopositioning of marine mammals using stereo photogrammetry". In: *2012 Oceans*. IEEE, pp. 1–6.

Hubel, David H and Torsten N Wiesel (1968). "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of physiology* 195.1, pp. 215–243.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International Conference on Machine Learning*, pp. 448–456.

Ji, Shuiwang et al. (2010). "3D convolutional neural networks for human action recognition". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. Citeseer, pp. 495–502.

Kelly, Marcella J (2001). "Computer-aided photograph matching in studies using individual identification: an example from Serengeti cheetahs". In: *Journal of Mammalogy* 82.2, pp. 440–449.

Kim, Sung and Riley Casper (2013). "Applications of convolution in image processing with MATLAB". In:

Kingma, Diederik P and Jimmy Ba (2014). "Adam: {A} Method for Stochastic Optimization". In: *CoRR* abs/1412.6. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

Koch, Gregory (2015). "Siamese neural networks for one-shot image recognition". In:

Kohler, Nancy E and Patricia A Turner (2001). "Shark tagging: a review of conventional methods and studies". In: *The behavior and sensory biology of elasmobranch fishes: an anthology in memory of Donald Richard Nelson*. Springer, pp. 191–224.

Körschens, Matthias, Björn Barz, and Joachim Denzler (2018). "Towards automatic identification of elephants in the wild". In: *arXiv preprint arXiv:1812.04418*.

Koziarski Michałand Cyganek, Bogusław (2018). "Impact of low resolution on image recognition with deep neural networks: An experimental study". In: *International Journal of Applied Mathematics and Computer Science* 28.4.

Kristan, Matej et al. (2017). "The visual object tracking vot2017 challenge results". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1949–1972.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

Kukačka, Jan, Vladimir Golkov, and Daniel Cremers (2017). "Regularization for deep learning: A taxonomy". In: *arXiv preprint arXiv:1710.10686*.

Lan, Qiang et al. (2017). "High performance implementation of 3d convolutional neural networks on a gpu". In: *Computational intelligence and neuroscience* 2017.

Langtimm, Catherine A et al. (2004). "Survival estimates for Florida manatees from the photo-identification of individuals". In: *Marine Mammal Science* 20.3, pp. 438–463.

LaRue, Michelle A, Seth Stapleton, and Morgan Anderson (2017). "Feasibility of using high-resolution satellite imagery to assess vertebrate wildlife populations". In: *Conservation biology* 31.1, pp. 213–220.

Lavy, Adi et al. (2015). "A quick, easy and non-intrusive method for underwater volume and surface area evaluation of benthic organisms by 3D computer modelling". In: *Methods in Ecology and Evolution* 6.5, pp. 521–531.

Lebreton, Jean-Dominique et al. (1992). "Modeling survival and testing biological hypotheses using marked animals: a unified approach with case studies". In: *Ecological monographs* 62.1, pp. 67–118.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *nature* 521.7553, p. 436.

LeCun, Yann et al. (1998). "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, pp. 9–50.

Levy, Keren, Amit Lerner, and Nadav Shashar (2014). "Mate choice and body pattern variations in the Crown Butterfly fish Chaetodon paucifasciatus (Chaetodontidae)". In: *Biology open* 3.12, pp. 1245–1251.

Li, Xiang et al. (2018). "Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift". In: *CoRR* abs/1801.0. arXiv: 1801.05134. URL: http://arxiv.org/abs/1801.05134.

Lin, Tsung-Yi et al. (2014). "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer, pp. 740–755.

Lin, Tsung-Yi et al. (2015). "Learning deep representations for ground-to-aerial geolocalization". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5007–5015.

Lowe, David G (2004). "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2, pp. 91–110.

Lytle, David A et al. (2010). "Automated processing and identification of benthic invertebrate samples". In: *Journal of the North American Benthological Society* 29.3, pp. 867–874.

Mahsereci, Maren et al. (2017). "Early stopping without a validation set". In: *arXiv preprint arXiv:1703.09580*.

Martínez-Jauregui, María et al. (2012). "Population resilience of the Mediterranean monk seal Monachus monachus at Cabo Blanco peninsula". In: *Marine Ecology Progress Series* 461, pp. 273–281.

Matthé, Maximilian et al. (2017). "Comparison of photo-matching algorithms commonly used for photographic capture–recapture studies". In: *Ecology and evolution* 7.15, pp. 5861–5872.

McMahon, C R, C J A Bradshaw, and G C Hays (2006). "Branding can be justified in vital conservation research". In: *Nature* 439.7075, p. 392.

Measey, J et al. (2014). "Cape collaborations for amphibian solutions". In: *FrogLog* 22, pp. 46–47.

Mikołajczyk, Agnieszka and Michał Grochowski (2018). "Data augmentation for improving deep learning in image classification problem". In: *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE, pp. 117–122.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Nordhausen, Klaus (2009). "The elements of statistical learning: data mining, inference, and prediction, by Trevor Hastie, Robert Tibshirani, Jerome Friedman". In: *International Statistical Review* 77.3, p. 482.

Olsen, Aaron M and Mark W Westneat (2015). "StereoMorph: an R package for the collection of 3D landmarks and curves using a stereo camera set-up". In: *Methods in Ecology and Evolution* 6.3, pp. 351–356.

Parkhi, Omkar M et al. "Deep face recognition." In:

Pellitteri-Rosa, Daniele et al. (2010). "Photographic identification in reptiles: a matter of scales". In: *Amphibia-Reptilia* 31.4, pp. 489–502.

Pennekamp, Frank and Nicolas Schtickzelle (2013). "Implementing image analysis in laboratory-based experimental systems for ecology and evolution: a hands-on guide". In: *Methods in Ecology and Evolution* 4.5, pp. 483–492.

Perez, Luis and Jason Wang (2017). "The effectiveness of data augmentation in image classification using deep learning". In: *arXiv preprint arXiv:1712.04621*.

Prechelt, Lutz (1998). "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, pp. 55–69.

Provost, Foster J et al. (1998). "The case against accuracy estimation for comparing induction algorithms." In: *ICML*. Vol. 98, pp. 445–453.

Qian, Ning (1999). "On the momentum term in gradient descent learning algorithms". In: *Neural networks : the official journal of the International Neural Network Society* 12, pp. 145–151.

Rahman, Ziaur et al. (2019). "A framework for fast automatic image cropping based on deep saliency map detection and gaussian filter". In: *International Journal of Computers and Applications* 41.3, pp. 207–217.

Redmon, Joseph et al. (2016). "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Ren, Shaoqing et al. (2015). "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*, pp. 91–99.

Rezatofighi, Hamid et al. (2019). *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*. arXiv: 1902.09630 [cs.CV].

Rosenblatt, Frank (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

Saunders, Luke J, Richard A Russell, and David P Crabb (2012). "The coefficient of determination: what determines a useful R2 statistic?" In: *Investigative ophthalmology & visual science* 53.11, pp. 6830–6832.

Schneider, Stefan et al. (2019). "Past, present and future approaches using computer vision for animal re-identification from camera trap data". In: *Methods in Ecology and Evolution* 10.4, pp. 461–470.

Schwarz, Carl J and George A F Seber (1999). "Estimating animal abundance: review III". In: *Statistical Science*, pp. 427–456.

Simonyan, Karen and Andrew Zisserman (2014a). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

– (2014b). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1. arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

Smith, Leslie N (2015). "No More Pesky Learning Rate Guessing Games". In: *CoRR* abs/1506.0. arXiv: 1506.01186. URL: http://arxiv.org/abs/1506.01186.

Speed, Conrad W, Mark G Meekan, and Corey J A Bradshaw (2007). "Spot the match–wildlife photo-identification using information theory". In: *Frontiers in zoology* 4.1, p. 2.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1, pp. 1929–1958. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2627435.2670313.

Stoddard, Mary Caswell, Rebecca M Kilner, and Christopher Town (2014). "Pattern recognition algorithm reveals how birds evolve individual egg pattern signatures". In: *Nature communications* 5, p. 4117.

Stoddard, Mary Caswell et al. (2016). "Camouflage and clutch survival in plovers and terns". In: *Scientific reports* 6, p. 32059.

Sutskever, Ilya et al. (2013). "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*, pp. 1139–1147.

Swanson, Alexandra et al. (2015). "Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna". In: *Scientific data* 2, p. 150026.

Szegedy, Christian, Alexander Toshev, and Dumitru Erhan (2013). "Deep neural networks for object detection". In: *Advances in neural information processing systems*, pp. 2553–2561.

Szegedy, Christian et al. (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

T. Bolger, Douglas et al. (2012). "A computer-assisted system for photographic mark-recapture analysis". In: *Methods in Ecology and Evolution* 3, pp. 813–822. DOI: 10.1111/j.2041-210X.2012.00212.x.

Taigman, Yaniv et al. (2014). "Deepface: Closing the gap to human-level performance in face verification". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708.

Tan, Chuanqi et al. (2018). "A survey on deep transfer learning". In: *International Conference on Artificial Neural Networks*. Springer, pp. 270–279.

Tankus, Ariel and Yehezkel Yeshurun (2008). "Computer vision, camouflage breaking and countershading". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 364.1516, pp. 529–536.

Taylor, Luke and Geoff Nitschke (2017). "Improving Deep Learning using Generic Data Augmentation". In: *CoRR* abs/1708.0. arXiv: 1708.06020. URL: http://arxiv.org/abs/1708.06020.

Tieleman, T and G Hinton. "RMSprop Gradient Optimization". In: URL: http://www.cs.toronto.edu/{~}tijmen/csc321/slides/lecture{_}slides{_}lec6.pdf.

Villon, Sébastien et al. (2016). "Coral reef fish detection and recognition in underwater videos by supervised machine learning: Comparison between Deep Learning and HOG+ SVM methods". In: *International Conference on Advanced Concepts for Intelligent Vision Systems*. Springer, pp. 160–171.

Watkins, William A et al. (1993). "Sperm whales tagged with transponders and tracked underwater by sonar". In: *Marine mammal science* 9.1, pp. 55–67.

Weinstein, Ben G (2018). "A computer vision for animal ecology". In: *journal of animal ecology* 87.3, pp. 533–545.

Weisstein, Eric W (2004). "Affine transformation". In:

Whitehead, Hal, Jenny Christal, and Peter L Tyack (2000). "Studying cetacean social structure in space and time". In: *Cetacean societies: Field studies of dolphins and whales*, pp. 65–86.

Widrow, Bernard and Michael A Lehr (1990). "30 years of adaptive neural networks: perceptron, madaline, and backpropagation". In: *Proceedings of the IEEE* 78.9, pp. 1415–1442.

Wilson, Rory P and Clive R McMahon (2006). "Measuring devices on wild animals: what constitutes acceptable practice?" In: *Frontiers in Ecology and the Environment* 4.3, pp. 147–154.

Xu, Bing et al. (2015). "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853*.

Zagoruyko, Sergey and Nikos Komodakis (2015). "Learning to compare image patches via convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4353–4361.

Zbontar, Jure and Yann LeCun (2015). "Computing the stereo matching cost with a convolutional neural network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1592–1599.

Zhou, Bolei et al. (2017). "Scene parsing through ade20k dataset". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 633–641.

Zhu, Xiaojin and Andrew B Goldberg (2009). "Introduction to semi-supervised learning". In: *Synthesis lectures on artificial intelligence and machine learning* 3.1, pp. 1–130.

Zhu, Xiaojin Jerry (2005). *Semi-supervised learning literature survey*. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences.